IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	Mitchell Askenas	Atty Dkt:	1436/159
Serial No:	09/848,812	Art Unit:	2623
Date Filed:	May 4, 2001	Examiner:	Saltarelli, D.

Invention: Method and Apparatus for A Cable TV Server

DECLARATION UNDER 37 C.F.R. §1.131

1. We are co-inventors of the Method and Apparatus for a Cable TV Server that is the subject of the above-identified patent application.
2. We invented and created original source code prior to December 15, 1999. The code name used for this project at that time was Arctos.
3. The version control software used by our company was called Source Safe. Source Safe allows us as software developers to check out a file, make changes, and then check the file back in. SourceSafe is a version control system that allows back-tracking to previous versions of a file or the entire projects. To facilitate back-tracking and control of versions, SourceSafe maintains file change histories, audit trail logs, and disaster recovery for source code files. If files are

added or modified, Source Safe records will show it. Accuracy of Source Safe date stamps on source code files are guaranteed by its maker, Microsoft.

4. Prior to December 15, 1999, the Arctos software was demonstrated in conjunction with a headend simulated by a Vivid™ Simulator acting as settop box code downloaders and Motorola DCCG DigiCable Control Channel Generator as channel map downloader, thereby demonstrating that the invention of the above-identified application had been reduced to practice.
5. The project was active in the December 1999 time frame and much of the software was checked out to developers who sought to make improvements that might make the system more marketable. While the exact state of the software in the combination of software modules used to demonstrate thc invention's capabilities prior to December 15, 1999 can not be precisely ascertained from the Source Safe records, the versions checked in shortly thereafter have been preserved.
6. Relevant excerpts of the software dated prior to December 15, 1999 or shortly thereafter have been attached hereto as exhibits. Dates prior to December 15, 1999 have been redacted to maintain their confidentiality.
7. In particular, with respect to claim 1, prior to December 15, 1999, the Arctos software included as part of the Carousel Explorer a browser application that produced a display of a web page. The browser application was called CarouselExplorerView.cpp and its source code as of prior to December 15, 1999 is attached hereto as Exhibit A.
8. Further prior to December 15, 1999, the Arctos software included an image capture module that captured successive images of the web page displayed by the

browser application. The image capture module was part of the Carousel Explorer. Its source code in its form when checked in on December 26, 1999 called ScreenImage.cpp is attached hereto as Exhibit B.

9. Further prior to December 15, 1999, the image compressor that compressed the successive images captured by the image capture module of the Arctos software was called CarToMux.cpp. The source code for CarToMux .cpp in its form when checked in on December 23, 1999 is attached hereto as Exhibit C.

10. Further prior to December 15, 1999, compressed images from the image compressor were assigned a transport control protocol internet protocol (TCP/IP) port number by ChannelMgr. Each port was associated with a corresponding television channel. Source code for the ChannelMgr. in its form when checked in December 31, 1999 is attached hereto as Exhibit D.

11. Further prior to December 15, 1999, the TCP/IP numbered streams were delivered by a multiplexer as separately selectable television channels thereby permitting simultaneous viewing at any of the subscriber televisions at which the channel is selected. The multiplexor was implemented prior to December 15, 1999 and a multiplexor source code file from prior to December 15, 1999 is attached hereto as Exhibit E.

12. By December 15, 1999, Arctos software had been used to demonstrate a working embodiment of the web content server that produced a television channel containing images captured from a browser for viewing by any television connected to receive the television signal.

13. The conception of the invention is documented in the attached Exhibit F entitled Arctos Version 1.0 Functional Specification ("the Spec") and bearing document

false statements may jeopardize the validity of the application in connection with which this declaration is being submitted to the Patent and Trademark Office, or any patent issued thereon.

October 24, 2007
Date

Mitchell S. Askenas

Date

Airan Landau

Date

Allan Moluf

Date

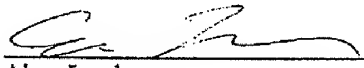
Elena Y. Pavlovskaia

Date

Robert B. Sigmon, Jr.

01436/00159 726897.1

false statements may jeopardize the validity of the application in connection with which this declaration is being submitted to the Patent and Trademark Office, or any patent issued thereon.

Date	Mitchell S. Askenas
10/1/2007	
Date	Airan Landau
Date	Allan Moluf
Date	Elena Y. Pavlovskaia
Date	Robert B. Sigmon, Jr.

01436/00159 726897.1

false statements may jeopardize the validity of the application in connection with which this declaration is being submitted to the Patent and Trademark Office, or any patent issued thereon.

Date	Mitchell S. Askenas
Date	Airan Landau
2007 Oct 01 Date	Allan Moluf
Date	Elena Y. Pavlovskaia
Date	Robert B. Sigmon, Jr.

01436/00159 726897.1

false statements may jeopardize the validity of the application in connection with which this declaration is being submitted to the Patent and Trademark Office, or any patent issued thereon.

Date	Mitchell S. Askenas
Date	Airan Landau
Date	Allan Moluf
10.01.2007 Date	Elena Y. Pavlovskaia
9-26-2007 Date	Robert B. Sigmon, Jr.

01436/00159 726897.1

CarouselExplorerView.cpp

```
// CarouselExplorerView.cpp : implementation of the CChannelView class
//

#include "stdafx.h"
#include "CarouselExplorer.h"
#include "CarouselExplorerDefs.h"

#include "CarouselExplorerDoc.h"
#include "CarouselExplorerView.h"

#include "MainFrm.h"

#include <atlbase.h>
#include "mshtml.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CChannelView

IMPLEMENT_DYNCREATE(CChannelView, CFormView)

BEGIN_MESSAGE_MAP(CChannelView, CFormView)
        //{{AFX_MSG_MAP(CChannelView)
        ON_WM_WINDOWPOSCHANGING()
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CChannelView construction/destruction

CChannelView::CChannelView()
        : CFormView(CChannelView::IDD)
{
        //{{AFX_DATA_INIT(CChannelView)
        m_strDescription = _T("");
        //}}AFX_DATA_INIT
        // TODO: add construction code here

        m_lpDispDoc = NULL;
}

CChannelView::~CChannelView()
{
}

void CChannelView::DoDataExchange(CDataExchange* pDX)
{
        CFormView::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CChannelView)
        DDX_Control(pDX, IDC_CHANNEL_DESCRIPTION, m_wndDescription);
        DDX_Control(pDX, IDC_EXPLORER, m_wndBrowser);
        DDX_Text(pDX, IDC_CHANNEL_DESCRIPTION, m_strDescription);
        //}}AFX_DATA_MAP
}

BOOL CChannelView::PreCreateWindow(CREATESTRUCT& cs)
{
        cs.style &= ~WS_OVERLAPPEDWINDOW;
```

```
        cs.style &= ~WS_BORDER;
        cs.style &= ~WS_DLGFRAME;
        cs.style &= ~DS_3DLOOK;
        cs.style &= ~0x8000;
        cs.dwExStyle &= ~WS_EX_WINDOWEDGE;

        return CFormView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CChannelView diagnostics

#ifdef _DEBUG
void CChannelView::AssertValid() const
{
        CFormView::AssertValid();
}

void CChannelView::Dump(CDumpContext& dc) const
{
        CFormView::Dump(dc);
}

CChannelDoc* CChannelView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CChannelDoc)));
        return (CChannelDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CChannelView message handlers

BEGIN_EVENTSINK_MAP(CChannelView, CFormView)
    //{{AFX_EVENTSINK_MAP(CChannelView)
        ON_EVENT(CChannelView, IDC_EXPLORER, 250 /* BeforeNavigate2 */,
OnBeforeNavigate2, VTS_DISPATCH VTS_PVARIANT VTS_PVARIANT VTS_PVARIANT VTS_PVARIANT
VTS_PVARIANT VTS_PBOOL)
        ON_EVENT(CChannelView, IDC_EXPLORER, 108 /* ProgressChange */,
OnProgressChange, VTS_I4 VTS_I4)
        ON_EVENT(CChannelView, IDC_EXPLORER, 259 /* DocumentComplete */,
OnDocumentComplete, VTS_DISPATCH VTS_PVARIANT)
        ON_EVENT(CChannelView, IDC_EXPLORER, 112 /* PropertyChange */,
OnPropertyChangeExplorer, VTS_BSTR)
        ON_EVENT(CChannelView, IDC_EXPLORER, 252 /* NavigateComplete2 */,
OnNavigateComplete2, VTS_DISPATCH VTS_PVARIANT)
        //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

void CChannelView::OnBeforeNavigate2(LPDISPATCH pDisp, VARIANT FAR* URL, VARIANT
FAR* Flags, VARIANT FAR* TargetFrameName, VARIANT FAR* PostData, VARIANT FAR*
Headers, BOOL FAR* Cancel)
{
        *Cancel = !GetDocument()->BeforeNavigate(CString(URL->bstrVal));
}

void CChannelView::OnProgressChange(long Progress, long ProgressMax)
{
//      TRACE("After %d %d   %s\n", Progress, ProgressMax, strUrl);
}

void CChannelView::OnInitialUpdate()
{
```

```
        CFormView::OnInitialUpdate();
}

void CChannelView::OnDocumentComplete(LPDISPATCH pDisp, VARIANT FAR* URL)
{

/*      if (m_lpDispDoc && m_lpDispDoc == pDisp)
        {
                m_lpDispDoc = NULL;
*/
/*      LPUNKNOWN lpUnknown;
        LPUNKNOWN lpUnknownWB = NULL;
        LPUNKNOWN lpUnknownDC = NULL;

        if (NULL != (lpUnknown = m_wndBrowser.GetControlUnknown()))
        {
                if (SUCCEEDED(lpUnknown->QueryInterface(IID_IUnknown,
(LPVOID*)&lpUnknownWB)))
                {
                        if (SUCCEEDED(pDisp->QueryInterface(IID_IUnknown,
(LPVOID*)&lpUnknownDC)) &&
                                (lpUnknownWB == lpUnknownDC))
                        {
                                lpUnknownDC->Release();
*/
                                // Get the real document name from the document
object
                                CComPtr<IHTMLDocument2> i_doc =
(IHTMLDocument2*)m_wndBrowser.GetDocument();

                                CString strUrl("res://");
                                if (i_doc != NULL)
                                {
                                        CComBSTR bstrUrl;
                                        if (SUCCEEDED(i_doc->get_URL(&bstrUrl)))
                                                strUrl = bstrUrl;
                                }

                                GetDocument()->SetCurrentUrl(strUrl);
/*                      }
                        lpUnknownWB->Release();
                }
        }
*/
//      }
}

void CChannelView::OnPropertyChangeExplorer(LPCTSTR szProperty)
{
/*      TRACE("Property : '%s'\n", szProperty);

        VARIANT var = m_wndBrowser.GetProperty_(szProperty);
        */
}


void CChannelView::OnWindowPosChanging(WINDOWPOS FAR* lpwndpos)
{
        CFormView::OnWindowPosChanging(lpwndpos);

        // Stretch the browser control to the window
        if ((HWND)m_wndBrowser)
        {
```

```
                m_wndBrowser.MoveWindow(-2, -2, lpwndpos->cx + 4,
theApp.m_ScreenSize.cy + 4);
                m_wndDescription.MoveWindow(0, theApp.m_ScreenSize.cy + 4,

lpwndpos->cx, lpwndpos->cy - theApp.m_ScreenSize.cy - 4);
        }
}

void CChannelView::OnNavigateComplete2(LPDISPATCH pDisp, VARIANT FAR* URL)
{
        if (!m_lpDispDoc)
                m_lpDispDoc = pDisp;

}
```

ScreenImage.cpp

```
// ScreenImage.cpp: implementation of the CScreenImage class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "ScreenImage.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CScreenImage::CScreenImage(int CapWidth, int CapHeight)
: m_SrcDC(CWnd::GetDesktopWindow())
{
        m_CapWidth = CapWidth;
        m_CapHeight = CapHeight;

        m_TotalFrames = 0;
        m_TotalTime = 0;

    m_hDibSect = NULL;
    m_hOldDibSect = NULL;

/*      m_pDibSectBits = 0;
        m_BitsPerPixel = 0;
        m_PixelBufLen = 0;
    m_BytesPerPixel = 0;
        m_CapWidth = 0;
        m_CapHeight = 0;
    m_DibSect555 = FALSE;        // TRUE if 15-bit RGB mode
        */
}

CScreenImage::~CScreenImage()
{
        if (m_hOldDibSect)
                m_DibSectMemDC.SelectObject(m_hOldDibSect);

        if (m_hDibSect)
                ::DeleteObject(m_hDibSect);
}

BOOL CScreenImage::Init()
{
        // Get information on the system display
    int planes          = m_SrcDC.GetDeviceCaps(PLANES);
    m_BitsPerPixel  = m_SrcDC.GetDeviceCaps(BITSPIXEL) * planes;        // 8, 16,
24, 32
    m_BytesPerPixel = m_BitsPerPixel / 8;
        // 1, 2,  3,  4
    m_PixelBufLen   = m_BytesPerPixel * m_CapWidth * m_CapHeight;

    UINT rastercaps = m_SrcDC.GetDeviceCaps(RASTERCAPS);
    UINT palettized = rastercaps & RC_PALETTE;
    if (palettized)
        {
        TRACE("Palettized display cannot be used with MPEG encoder!");
```

```
            return FALSE;
    }

        // Create the memory dc to draw into
        m_DibSectMemDC.CreateCompatibleDC(&m_SrcDC);

        // Initialize the structure with the DIB parameters
    ::ZeroMemory(&m_DibSectBmpInfo, sizeof(DSBITMAPINFO));
    m_DibSectBmpInfo.bmiHeader.biSize                   = sizeof(BITMAPINFOHEADER);
    m_DibSectBmpInfo.bmiHeader.biWidth                  = m_CapWidth;
    m_DibSectBmpInfo.bmiHeader.biHeight                 = -m_CapHeight; // negative
specifies top-down dib
    m_DibSectBmpInfo.bmiHeader.biPlanes                 = 1;
    m_DibSectBmpInfo.bmiHeader.biBitCount               = m_BitsPerPixel;
    m_DibSectBmpInfo.bmiHeader.biCompression    = BI_BITFIELDS;
    m_DibSectBmpInfo.bmiHeader.biSizeImage              = 0;
    m_DibSectBmpInfo.bmiHeader.biXPelsPerMeter  = 0;
    m_DibSectBmpInfo.bmiHeader.biYPelsPerMeter  = 0;
    m_DibSectBmpInfo.bmiHeader.biClrUsed                = 0;
    m_DibSectBmpInfo.bmiHeader.biClrImportant   = 0;

    m_hDibSect = NULL;

    // Only create a dib for 16 bit color mode
        if (m_BytesPerPixel == 2)
        {
        // for 16-bit 5-6-5, using BI_BITFIELDS
        m_DibSectBmpInfo.r = 0x00F800;
        m_DibSectBmpInfo.g = 0x0007E0;
        m_DibSectBmpInfo.b = 0x00001F;
        //
        //  Attempt to create 5-6-5 format Dibsection.
        //
        m_hDibSect = CreateDIBSection(0, (BITMAPINFO *)&m_DibSectBmpInfo,
                DIB_RGB_COLORS, (void **)(&m_pDibSectBits), NULL, NULL);

        if (m_hDibSect == NULL)
                {
            //
            //  Failed to create 5-6-5.  Try 5-5-5.
            //
            //  NOTE: To maintain color integrity, current buffer must be
            //  treated as upside-down from normal, so *positive* number must
            //  be given as buffer Y-dim, and encoder's 'flip' flag is set.
            //
            m_DibSectBmpInfo.r = 0x007C00;
            m_DibSectBmpInfo.g = 0x0003E0;
            m_DibSectBmpInfo.b = 0x00001F;

            m_DibSectBmpInfo.bmiHeader.biHeight = m_CapHeight;
            //
            //  Attempt to create 5-5-5 format Dibsection.
            //  mdg:!!!! NOTE: This may benchmark much slower than 5-6-5.
            //
            m_hDibSect = CreateDIBSection(0, (BITMAPINFO *)&m_DibSectBmpInfo,
                    DIB_RGB_COLORS, (void **)(&m_pDibSectBits), NULL, NULL);
        }
    }
    if (m_hDibSect == NULL)
        {
        //
        //  Higher bit resolutions (24, 32) init here.
        //  16-bit create also gets one more chance if prev creates failed.
```

```
    //
    //  for 24/32-bit, using BI_BITFIELDS
    //  m_DibSectBmpInfo.r = 0xFF0000;
    //  m_DibSectBmpInfo.g = 0x00FF00;
    //  m_DibSectBmpInfo.b = 0x0000FF;
    //
    m_DibSectBmpInfo.bmiHeader.biCompression = BI_RGB;
    //
    //  NOTE for 16-bit: To maintain color integrity, current buffer must
    //  be treated as upside-down from normal, so *positive* number must
    //  be given as buffer Y-dim, and encoder's 'flip' flag is set.
    //
    if (m_BytesPerPixel == 2)
            {
        m_DibSectBmpInfo.bmiHeader.biHeight = m_CapHeight;
    }
    //
    //  Create dibsection for 32-bit, 24-bit, or 15-bit (5-5-5).
    //
    m_hDibSect = CreateDIBSection(0, (BITMAPINFO *)&m_DibSectBmpInfo,
            DIB_RGB_COLORS, (void **)(&m_pDibSectBits), NULL, NULL);
}
if (m_hDibSect == NULL)
    {
    //
    //  Failed to create DibSection!
    //
    DWORD ErrVal = GetLastError();
    TRACE("Error!  Dibsection picture buffer failed to initialize.");
            return FALSE;
}
    else
    {
    //
    //  DibSection created successfully.  Select into MemDC.
    //
    m_hOldDibSect = (HBITMAP)m_DibSectMemDC.SelectObject(m_hDibSect);
    //
    //  If 15/16 bit format, find actual RGB mask (5-5-5 vs 5-6-5).
    //
    m_DibSect555 = FALSE;
    if (m_BytesPerPixel == 2)
            {
        //
        //  Local struct gets pixel info.
        //
        struct LocalBMI
                    {
            BITMAPINFOHEADER bi;
            union
                        {
                RGBQUAD colors[256];
                DWORD fields[256];
            };
        };

        LocalBMI pix_info;

        ZeroMemory(&pix_info, sizeof(pix_info));
        pix_info.bi.biSize = sizeof(pix_info.bi);
        pix_info.bi.biBitCount = 0;
        //
        //  First call fills in param fields.
```

```
            //  Second call fills in mask fields.
            //
            GetDIBits(m_DibSectMemDC, m_hDibSect, 0, 1, NULL,
                    (BITMAPINFO *)&pix_info, DIB_RGB_COLORS);
            GetDIBits(m_DibSectMemDC, m_hDibSect, 0, 1, NULL,
                    (BITMAPINFO *)&pix_info, DIB_RGB_COLORS);

            if (pix_info.bi.biCompression == 0)
                    {
                if (pix_info.bi.biBitCount > 8)
                            {
                    if (pix_info.bi.biBitCount==16)
                        m_DibSect555 = TRUE;
                    else
                                    {
                                            TRACE("More than 16 bits.  Should
not get here\n");
                                return FALSE;
                                    }
                }
                                else
                                {
                                        TRACE("palettized.  Should not get here\n");
                                        return FALSE;
                }
            }
                        else if (pix_info.bi.biCompression == BI_BITFIELDS)
                        {
                DWORD allmasks = pix_info.fields[0] | pix_info.fields[1] |
pix_info.fields[2];
                if (allmasks == 0x7FFF)
                    m_DibSect555 = TRUE;
            }
            if (m_DibSect555 == TRUE)
                        {               // mdg:!!!!!!!!!!!!!
                TRACE("Video has init'd to 5-5-5 format! Capture functions may be
much slower as a result.");
                }
        }
    }

        return TRUE;
}

char* CScreenImage::GetScreenImage(int SrcX, int SrcY, int Width, int Height, int
DestX, int DestY)
{
//      DWORD dwStart = GetTickCount();
/*      LARGE_INTEGER start;
        LARGE_INTEGER end;
        LARGE_INTEGER freq;

        QueryPerformanceCounter(&start);
*/
    //------------------------------------------------
    //  Capture target area.-- DIBSection
    //
    m_DibSectMemDC.FillSolidRect(0, 0, m_CapWidth, m_CapHeight, RGB(0,0,0));

    if (!m_DibSectMemDC.BitBlt(DestX, DestY, Width, Height, &m_SrcDC, SrcX, SrcY,
SRCCOPY))
        {
                TRACE("Error %d\n", GetLastError());
```

```
                return NULL;
        }

/*      m_TotalFrames++;
//      m_TotalTime += GetTickCount() - dwStart;
        QueryPerformanceCounter(&end);
        QueryPerformanceFrequency(&freq);

        m_TotalTime += (DWORD)((end.QuadPart - start.QuadPart) * 1000 /
(double)freq.QuadPart);

    //
    //  Imprint cursor image on captured image.
    //
        POINT pt;
        GetCursorPos(&pt);

        HWND hWnd = ::WindowFromPoint(pt);

        if (hWnd)
        {
                HCURSOR hCursor = NULL;
                DWORD pid;
                DWORD tid = GetWindowThreadProcessId(hWnd, &pid);
                if (tid != GetCurrentThreadId())
                {
                        if (AttachThreadInput(GetCurrentThreadId(), tid, TRUE))
                        {
                                hCursor = GetCursor();
                                if (!AttachThreadInput(GetCurrentThreadId(), tid,
FALSE))
                                        ASSERT(FALSE);
                        }
                }

                if (!hCursor)
                        hCursor = GetCursor();

/*              ICONINFO ii;
                if (GetIconInfo(hCursor, &ii))
                {
                        pt.x -= ii.xHotspot;
                        pt.y -= ii.yHotspot;
                }
/
                m_DibSectMemDC.DrawIcon(pt.x, pt.y, hCursor);
        }
*/
        return m_pDibSectBits;
}


int CScreenImage::GetSize()
{
        return m_PixelBufLen;
}
```

CarToMux.c

```
/*-------------------------------------------------------------
| NAME: CarToMux.c
|-------------------------------------------------------------
|
| PURPOSE: To define the interface between the Arctos Carousel
|          process and the shared memory input to the Mux.
|
| DESCRIPTION: The Carousel process is concerned with
| creating and controlling shared memory input channels to
| the Mux and with supplying image bitmaps for processing by
| the Mux.
|
| The Mux is responsible for converting the bitmap into MPEG
| and sending the resulting frames according to how the
| channel is currently configured.
|
| This file defines how the Carousel process controls the
| Mux input.
|
| All functions in this API are identified by the prefix
| 'Cm' which stands for "Carousel to Mux".
|
| NOTE:
|
| HISTORY: [redacted] Tim Lee.
|
| Copyright [redacted] by ICTV, Inc.  All rights reserved.
-------------------------------------------------------------*/

#include <wtypes.h>
#include <winbase.h>
#include <winerror.h>
#include <time.h>
#include <stdio.h>
#include "NumTypes.h"
#include "SharedMem.h"
#include "IctvMpeg.h"
#include "ligoslib.h"
#include "CarToMux.h"


// #define CM_DUMP_TO_FILE 1
                // Define this symbol for debugging to route output
                // to a data file instead of to shared memory.

/*-------------------------------------------------------------
| NAME: MuxSM
|-------------------------------------------------------------
|
| PURPOSE: To identify the handle to the shared memory area
|          for communicating mux-level commands.
|
| DESCRIPTION: This shared memory area is used for
| communicating with the Dynamic Manager of the mux.
|
| This shared memory area is opened by 'CmSetup()' which
| must be called before any other 'Cm' function is called.
|
| NOTE:
|
| HISTORY: [redacted] Tim Lee
|
-------------------------------------------------------------*/
```

```
SM*     MuxSM = 0;

/*-------------------------------------------------------------
| NAME: TheChannelList
|-------------------------------------------------------------
|
| PURPOSE: To refer to the list of all existing channels.
|
| DESCRIPTION: When channels are added they are prepended to
| this list and when they are deleted they are removed from
| this list.
|
| NOTE:
|
| HISTORY: [redacted] Tim Lee
|
-------------------------------------------------------------*/
CmChannel* TheChannelList = 0;

BOOL    IsAllOutputChannelsDisabled = 0;
        // Non-zero if output from all channels is disabled
        // or zero to allow individual channels to send
        // output according to their particular suspend/resume
        // status.
        //
        // Later as needed: gather this and any other
        // mux-specific attibutes into a mux status struct.

/*-------------------------------------------------------------
| NAME: cmOutputBuffer
|-------------------------------------------------------------
|
| PURPOSE: To provide a standard place to put Ligos output.
|
| DESCRIPTION: This buffer is allocated by CmSetup().
|
| NOTE: Eventually this buffer will be replaced by code that
|       directs Ligos output directly to a shared memory area.
|
| HISTORY: 12.15.99 Tim Lee
|
-------------------------------------------------------------*/
u8*     CmOutputBuffer     = 0;
u32 CmOutputBufferSize = 0;

/*-------------------------------------------------------------
| NAME: CmActivateAll
|-------------------------------------------------------------
|
| PURPOSE: To enable output on all input channels that were
|          suspended by a previous call to 'CmDeactivateAll()'.
|
| DESCRIPTION: Only those channels that are individually
| marked as active are enabled for output.
|
| Always returns S_OK.
|
| EXAMPLE:
|
|               Result = CmActivateAll();
| NOTE:
|
| ASSUMES:
```

```
| HISTORY: [redacted] Tim Lee
|          12.15.99 Revised to disable/enable all output at
|                   the mux level rather than channel by
|                   channel.
-------------------------------------------------------------*/
                    // OUT: Result status code:
                    //
                    //   S_OK
s32
CmActivateAll()
{
        BOOL            ok;
        MuxControl  C;

        // If the output channels are globally disabled.
        if( IsAllOutputChannelsDisabled )
        {
                // Fill in the mux control record for the activate.
                C.OpCode    = MUX_ACTIVATE;
                C.DebugLevel  = 0;

                // Send an activate packet to the mux.
                ok = CmSendMuxCommand( MuxSM, &C, 0, 0 );

                // Now output channels are not globally disabled.
                IsAllOutputChannelsDisabled = 0;
        }

        // Return success.
        return( S_OK );
}

/*-------------------------------------------------------------
| NAME: CmActivateChannel
|-------------------------------------------------------------
|
| PURPOSE: To enable output on an input channel to the Mux.
|
| DESCRIPTION:
|
| Returns the channel status, whether the channel is was
| removed (S_OK) or if the channel doesn't exist to be resumed
| (E_NO_CHANNEL).
|
| EXAMPLE:
|
|     Result =
|             CmActivateChannel(
|           "Bis",    // Prefix used to one group of channels
|                     // from another.
|                     //
|           207 );    // Virtual channel number.
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Tim Lee
|          [redacted] Finished.
-------------------------------------------------------------*/
                    // OUT: Result status code:
                    //
                    //   S_OK
```

```
                        //    E_NO_CHANNEL
s32
CmActivateChannel(
        s8*     Prefix,  // A three letter zero-terminated
                         // string used to identify a group of
                         // related channels.  eg. "Bis".
                         //
        u32     Channel )// A small decimal number used to
                         // identify a specific channel in a
                         // group defined by the prefix.
                         //
                         // This is a "virtual channel number"
                         // in GI's terms.
{
        CmChannel*      A;
        BOOL        ok;

        // Search the channel list for an existing channel with
    // the same prefix and virtual channel number.
    A = CmFindChannel( Prefix, Channel );

    // If the channel exists.
    if( A )
    {
        // If the channel is not active.
        if( A->IsActive == 0 )
        {
                // Mark this channel as active.
                A->IsActive = 1;

                // Send an activate packet to the channel.
                ok = CmSendChannelCommand(
                                A, CHANNEL_ACTIVATE, 0, 0 );

                // If the command was acknowledged.
                if( ok )
                {
                        // Return OK status.
                                return( S_OK );
                }
                else // Not acknowledged.
                {
                        // Return failure.
                        return( E_NO_CHANNEL );
                }
        }
        else // Channel is already active.
        {
                        // Return OK status.
                        return( S_OK );
                }
    }
    else // The channel doesn't exist.
    {
        // Return the error code.
        return( E_NO_CHANNEL );
    }
}

/*-------------------------------------------------------------
| NAME: CmAddChannel
|-------------------------------------------------------------
|
```

PURPOSE: To add a carousel input channel to the Mux.

DESCRIPTION: Create all the mux resources necessary to for a new channel but doesn't enable the output: output is enabled by calling 'CmActivateChannel()'.

Returns the memory location of the bitmap buffer where images should be placed for output. Also returns the channel status, whether the channel is new (S_OK), already existing (E_CHANNEL_EXISTS) or unable to be used (E_INVALID_CHANNEL).

EXAMPLE:

```
// Declare a channel specification structure.
    CmChannelSpec C;

// Fill in the channel specification structure.

// Assign a prefix used to distinguish one group
// of channels from another.
C.Prefix[0] = 'B';
C.Prefix[1] = 'i';
C.Prefix[2] = 's';
C.Prefix[3] = 0;

// Assign a virtual channel number.
C.Channel = 207;

// Assign the program ID number to be used in the
// PAT/PMT.
    C.Prog = 42;

// Set the encoding bit rate to 5 megabits.
C.EncodingBitRate = 5000000.0;

// Set the channel bit rate to 50K bits.
C.ChannelBitRate = 50000.0;

// Set the presentation frame rate to 29.97 frames
// per second.
    C.Hz = 29.97;

// Assign an ES PID number of 17.
C.PID = 17;

// Assign a PMT PID number of 18.
C.PMT = 18;

    // Set the pixel row count to 480 rows.
C.RowCount = 480;

    // Set the pixel column count to 640 columns.
C.ColCount = 640;

// Set the pixel type to the only type that Ligos
// supports.
    C.PixelType = cmRGB565;

// Add the channel to the mux.
    Result = CmAddChannel( &C );
```

NOTE:

```
|
| ASSUMES:
|
| HISTORY: [redacted] Tim Lee
|          [redacted] Added opening of shared memory area.
-------------------------------------------------------------*/
                        // OUT: Result status code:
                        //
                        //   S_OK
                        //   E_CHANNEL_EXISTS
                        //   E_INVALID_CHANNEL
s32
CmAddChannel( CmChannelSpec* C )
{
        CmChannel*      A;
        s8                      Name[30];
        u32                     BufferSize;
        BOOL            ok;
        MuxControl  X;

        // Search the channel list for an existing channel with
    // the same prefix and virtual channel number.
    A = CmFindChannel( C->Prefix, C->Channel );

    // If the channel already exists.
    if( A )
    {
        // Return the result code for an existing channel.
        return( E_CHANNEL_EXISTS );
    }

    // Validate the channel parameters here and reject
    // bad input with an error.
    if( CmIsGoodChannelSpec( C ) == 0 )
    {
        // Return the failure code.
        return( E_INVALID_CHANNEL );
    }

    //
    // At this point we have a valid channel add request
    // that doesn't duplicate any existing channel.
    //

    // Fill in a mux control record.
    X.OpCode          = CHANNEL_MAKE;
    X.InputUnitType   = SHM_ENC;
    X.Config[0]       = 0;
        X.Prefix[0]       = C->Prefix[0];
        X.Prefix[1]       = C->Prefix[1];
        X.Prefix[2]       = C->Prefix[2];
        X.Prefix[3]       = C->Prefix[3];
        X.Channel         = C->Channel;
        X.Prog            = C->Prog;
        X.EncodingBitRate = C->EncodingBitRate;
        X.ChannelBitRate  = C->ChannelBitRate;
        X.Hz              = C->Hz;
        X.PID             = C->PID;
        X.PMT             = C->PMT;
        X.DebugLevel      = 0;

        // Request the creation of a new ShmEnc encoder by
        // the Mux.
```

```
/*      ok = CmSendMuxCommand(
                        MuxSM,    // Address of an open shared memory
                                  // area read by the Dynamic Manager
                                  // of a mux or by one of the input
                                  // units (encoders) of the mux.
                                  //
                &X,               // Address of a mux control record.
                                  //
                0,                // Any other input data require by
                                  // the command. Use zero if there is
                                  // no other data.  THis data must be
                                  // in Ictv MPEG Packet format.
                                  //
                0 );              // How many bytes of additional data
                                                // should be appended to the channel
                                                // control record.
*/
        ok = 1; // For now the mux will have a pre-defined list of channels

        // If the mux could not add the new channel.
        if( ok == 0 )
        {
                // Return the error code.
                return( E_INVALID_CHANNEL );
        }

    // Make a new record for an existing channel.
    A = (CmChannel*) malloc( sizeof( CmChannel ) );

    // Copy the channel spec to the channel record.
    A->C = *C;

    // Mark the channel as inactive initially.
    A->IsActive = 0;

    // Make a name for the shared memory area by
    // combining the prefix with the channel number.
        sprintf( Name, "%s%d", C->Prefix, C->Channel );

        // Calculate the size of the shared memory area:
        // always use 128 K for now.
        BufferSize = 128 * 1024;

    // Make the shared memory area for the channel.
    A->S =
     OpenSM(
        Name,       // The name of the shared memory area,
                    // a zero-terminated ASCII string up
                    // to 250 bytes long.
                BufferSize );
                                    // The size of the shared memory area
                                    // in bytes.
    //
    // Prepend channel record to the list of all channels.
    //

        ResetEvent(A->S->DataEvent);

    // If there is already a channel in the list.
    if( TheChannelList )
    {
        // Change the prior link of the current first record
        // in the list to point to the new record.
```

```
        TheChannelList->Prior = A;
    }

    // Connect the forward link from A to the existing first
    // record in the list or zero if there is no other record.
    A->Next = TheChannelList;

    // Make 'A' the first record in the list.
    TheChannelList = A;
    A->Prior = 0;

        // Return the success code.
        return( S_OK );
}

/*-------------------------------------------------------------
| NAME: CmFindChannel
|-------------------------------------------------------------
|
| PURPOSE: To find an existing channel in the list of all
|          channels.
|
| DESCRIPTION:
|
| EXAMPLE:
|
|     C =
|             CmFindChannel(
|          "Bis",    // Prefix used to one group of channels
|                    // from another.
|                    //
|          207 );    // Virtual channel number.
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]Tim Lee
-------------------------------------------------------------*/
                    // OUT: The address of a CmChannel record
                    //      or zero if record not found.
                    //
CmChannel*
CmFindChannel(
        s8*     Prefix,  // A three letter zero-terminated
                         // string used to identify a group of
                         // related channels.  eg. "Bis".
                         //
        u32     Channel )// A small decimal number used to
                         // identify a specific channel in a
                         // group defined by the prefix.
                         //
                         // This is a "virtual channel number"
                         // in GI's terms.
{
        CmChannel* A;

        // Refer to the first channel record in the list.
        A = TheChannelList;

        // While there is a channel record.
        while( A )
        {
                // If the channel number matches the desired
```

```
                // channel.
                if( A->C.Channel == Channel )
                {
                        // If the prefix matches the desired prefix.
                        if( ( A->C.Prefix[0] == Prefix[0] ) &&
                            ( A->C.Prefix[1] == Prefix[1] ) &&
                            ( A->C.Prefix[2] == Prefix[2] ) )
                        {
                                // Return the address of the channel
                                // record.
                                return( A );
                        }
                }

                // Advance to the next record.
                A = A->Next;
        }

        // Record was not found. Return zero.
        return( 0 );
}

/*-------------------------------------------------------------
| NAME: CmDeactivateAll
|-------------------------------------------------------------
|
| PURPOSE: To disable output on all active input channels.
|
| DESCRIPTION: Only those channels that are currently active
| at the time of this call are will be enabled on a subsequent
| call to 'CmActivateAll()'.
|
| Always returns S_OK.
|
| EXAMPLE:
|
|               Result = CmDeactivateAll();
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Tim Lee
|          12.15.99 Revised to disable/enable all output at
|                   the mux level rather than channel by
|                   channel.
-------------------------------------------------------------*/
                        // OUT: Result status code:
                        //
                        //   S_OK
s32
CmDeactivateAll()
{
        BOOL            ok;
        MuxControl  C;

        // If the output channels are not globally disabled.
        if( IsAlloutputChannelsDisabled == 0 )
        {
                // Fill in the mux control record for the activate.
                C.OpCode      = MUX_DEACTIVATE;
                C.DebugLevel  = 0;

                // Send an activate packet to the mux.
```

```
                ok = CmSendMuxCommand( MuxSM, &C, 0, 0 );

                // Now all output is globally disabled.
                IsAllOutputChannelsDisabled = 1;
        }

        // Return success.
        return( S_OK );
}

/*-------------------------------------------------------------
| NAME: CmDeactivateChannel
|-------------------------------------------------------------
|
| PURPOSE: To disable output on a channel going into the Mux.
|
| DESCRIPTION:
|
| Returns the channel status, whether the channel is was
| removed (S_OK) or if the channel doesn't exist to be resumed
| (E_NO_CHANNEL).
|
| EXAMPLE:
|
|     Result =
|             CmDeactivateChannel(
|           "Bis",    // Prefix used to one group of channels
|                     // from another.
|                     //
|           207 );    // Virtual channel number.
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Tim Lee
-------------------------------------------------------------*/
                    // OUT: Result status code:
                    //
                    //   S_OK
                    //   E_NO_CHANNEL
s32
CmDeactivateChannel(
        s8*     Prefix,  // A three letter zero-terminated
                         // string used to identify a group of
                         // related channels.  eg. "Bis".
                         //
        u32     Channel )// A small decimal number used to
                         // identify a specific channel in a
                         // group defined by the prefix.
                         //
                         // This is a "virtual channel number"
                         // in GI's terms.
{
        CmChannel*      A;
        BOOL            ok;

        // Search the channel list for an existing channel with
    // the same prefix and virtual channel number.
    A = CmFindChannel( Prefix, Channel );

    // If the channel exists.
    if( A )
    {
```

```
        // If the channel is active.
        if( A->IsActive )
        {
                // Mark this channel as inactive.
                A->IsActive = 0;

                // Send an deactivate packet to the channel.
                ok = CmSendChannelCommand(
                                A, CHANNEL_DEACTIVATE, 0, 0 );

                // If the command was acknowledged.
                if( ok )
                {
                        // Return OK status.
                                return( S_OK );
                }
                else // Not acknowledged.
                {
                        // Return failure.
                        return( E_NO_CHANNEL );
                }
        }

                // Return OK status.
                return( S_OK );
    }
    else // The channel doesn't exist.
    {
        // Return the error code.
        return( E_NO_CHANNEL );
    }
}

/*-------------------------------------------------------------
| NAME: CmIsGoodChannelSpec
|-------------------------------------------------------------
|
| PURPOSE: To test if a channel specification record contains
|          values that are within bounds.
|
| DESCRIPTION: Returns 1 if all the fields within the channel
| specification are valid, else returns 0.
|
| NOTE:
|
| HISTORY: [redacted]Tim Lee
|
-------------------------------------------------------------*/
BOOL
CmIsGoodChannelSpec( CmChannelSpec* C )
{
        // If the three letter channel prefix is not an ASCII
        // letter or the string terminator is missing.
        if( !IsASCIILetter( C->Prefix[0] ) ||
        !IsASCIILetter( C->Prefix[1] ) ||
        !IsASCIILetter( C->Prefix[2] ) ||
        C->Prefix[3] != 0 )
        {
                // Return code for invalid.
                return( 0 );
        }

        // If the Channel number is out of bounds.
```

```
if( C->Channel < 1 || C->Channel > 4095 )
{
        // Return code for invalid.
        return( 0 );
}

// If the Program number is out of bounds.
if( C->Prog < 1 || C->Prog > 65535 )
{
        // Return code for invalid.
        return( 0 );
}

// If the encoding bit rate is out of bounds.
if( C->EncodingBitRate <  1000000.0 ||
C->EncodingBitRate > 10000000.0  )
{
        // Return code for invalid.
        return( 0 );
}

// If the channel bit rate is out of bounds.
if( C->ChannelBitRate <    10000.0 ||
C->ChannelBitRate > 10000000.0  )
{
        // Return code for invalid.
        return( 0 );
}

// If the output frame rate is invalid.
if( C->Hz != ((f64) 23.976) &&
    C->Hz != ((f64) 29.97) )
{
        // Return code for invalid.
        return( 0 );
}

// If the PID is out of bounds.
if( C->PID < 16 || C->PID > 8190 )
{
        // Return code for invalid.
        return( 0 );
}

// If the PMT is out of bounds.
if( C->PMT < 16 || C->PMT > 8190 )
{
        // Return code for invalid.
        return( 0 );
}

// If the pixel row count is not supported.
if( C->RowCount != 480 )
{
        // Return code for invalid.
        return( 0 );
}

// If the pixel column count is not supported.
if( C->ColCount != 640 )
{
        // Return code for invalid.
        return( 0 );
```

```
        }

        // If the pixel type is not supported.
        if( C->PixelType != cmRGB565 )
        {
                // Return code for invalid.
                return( 0 );
        }

        // If a pixel buffer is given but it has
        // no size.
        if( C->Buffer && C->DataSize == 0 )
        {
                // Return code for invalid.
                return( 0 );
        }

        // Otherwise, the record is valid.
        return( 1 );
}

/*-------------------------------------------------------------
| NAME: CmRemoveChannel
|-------------------------------------------------------------
|
| PURPOSE: To remove a carousel input channel from the Mux.
|
| DESCRIPTION: Stops transmitting data and deallocates all the
| resources assigned to the channel.
|
| Returns the channel status, whether the channel is was
| removed (S_OK) or if the channel doesn't exist to be removed
| (E_NO_CHANNEL).
|
| EXAMPLE:
|
|     Result =
|             CmRemoveChannel(
|           "Bis",    // Prefix used to one group of channels
|                     // from another.
|                     //
|           207 );    // Virtual channel number.
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Tim Lee
-------------------------------------------------------------*/
                    // OUT: Result status code:
                    //
                    //   S_OK
                    //   E_NO_CHANNEL
s32
CmRemoveChannel(
        s8*     Prefix,  // A three letter zero-terminated
                         // string used to identify a group of
                         // related channels.  eg. "Bis".
                         //
        u32     Channel )// A small decimal number used to
                         // identify a specific channel in a
                         // group defined by the prefix.
                         //
                         // This is a "virtual channel number"
```

```
                        // in GI's terms.
{
        CmChannel*      A;
        BOOL        ok;

        // Search the channel list for an existing channel with
    // the same prefix and virtual channel number.
    A = CmFindChannel( Prefix, Channel );

    // If the channel exists.
    if( A )
    {
                //
                // Extract the channel record from the channel list.
                //

                // If there is a record prior to A in the list.
                if( A->Prior )
                {
                        // Link the prior record to the record after A.
                        A->Prior->Next = A->Next;
                }

                // If there is a link after A in the list.
                if( A->Next )
                {
                        // Link the next record back to the record
                        // before A.
                        A->Next->Prior = A->Prior;
                }

                // If A is the first record in the list.
                if( TheChannelList == A )
                {
                        // Refer the list to the next record.
                        TheChannelList = A->Next;
                }

                // Request the deletion of an existing encoder
                // by the Mux.
                ok = CmSendChannelCommand(
                                A,
                                CHANNEL_DELETE,
                                0,
                                0 );

                // Close the shared memory area.
                CloseSM( A->S );

                // Free the channel record.
                free( A );

        // Report successful channel removal.
        return( S_OK );
    }
    else // The channel doesn't exist.
    {
        // Return the status code saying that there is no
        // channel with the given prefix and channel number.
        return( E_NO_CHANNEL );
    }
}
```

```
/*-------------------------------------------------------------
| NAME: CmSendChannelCommand
|-------------------------------------------------------------
|
| PURPOSE: To send a channel control command to a mux input
|          channel.
|
| DESCRIPTION: Returns one if the command was sent and
| acknowledged by a mux input channel or returns zero if
| the timeout period for the shared memory channel elapses.
|
| EXAMPLE:
|
|         ok = CmSendChannelCommand(
|                  A,
|                  CHANNEL_ACTIVATE,
|                  0,
|                  0 );
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Tim Lee from 'SendDataSM'.
-------------------------------------------------------------*/
BOOL
CmSendChannelCommand(
        CmChannel*      A,   // Address of an existing channel record.
                             //
        u32                  OpCode,
                             // Channel command code: see Mux control
                             // opcodes in "IctvMpeg.h".
                             //
        u8*                  OpData,
                             // Any other input data require by
                             // the command. Use zero if there is
                             // no other data.
                             //
        u32                  OpDataSize )
                                     // How many bytes of additional data
                                     // should be appended to the channel
                                     // control record.
{
        CmChannelSpec*  S;
        MuxControl           C;
        BOOL                 ok;
        u32             DataSize, PacketSize;
        SM*                          M;

        // Calculate the overall size of the data portion
        // of the Share Memory Packet.
        DataSize =
                sizeof( PacketHeaderIM )
                        // For the control packet header.
                +
            sizeof( MuxControl )
                // The size of the control packet data.
            +
            OpDataSize;
                // The size of any additional data tacked
                // on the end.

        // Calculate the total size of the Shared Memory
        // packet in bytes.
```

```
        PacketSize = sizeof( PacketHeaderSM ) + DataSize;

        // Refer to the shared memory area of the channel
        // as 'M'.
        M = A->S;

        // If the given data size exceeds the size of the
        // shared memory buffer.
        if( PacketSize > M->BufferSize )
        {
                // Return failure.
                return( 0 );
        }

        // Set the opcode field in the control record.
        C.OpCode = OpCode;

        // Treat the encoder type as shared memory.
        C.InputUnitType = SHM_ENC;

        // Clear the configuration string.
        C.Config[0] = 0;

        // Don't enable debuggin.
        C.DebugLevel = 0;

        // Refer to the channel spec record of the channel
        // record.
        S = &A->C;

        // Copy the channel specification to the
        // channel control record.
        C.Prefix[0]       = S->Prefix[0];
        C.Prefix[1]       = S->Prefix[1];
        C.Prefix[2]       = S->Prefix[2];
        C.Prefix[3]       = S->Prefix[3];
        C.Channel         = S->Channel;
        C.Prog            = S->Prog;
        C.EncodingBitRate = S->EncodingBitRate;
        C.ChannelBitRate  = S->ChannelBitRate;
        C.Hz              = S->Hz;
        C.PID             = S->PID;
        C.PMT             = S->PMT;

        // Send the data to the channel input buffer.
        ok = CmSendMuxCommand( M, &C, OpData, OpDataSize );

        // Return the status of the send.
        return( ok );
}

/*-------------------------------------------------------------
| NAME: CmSendMuxCommand
|-------------------------------------------------------------
|
| PURPOSE: To send a control command to a mux or input
|          channel.
|
| DESCRIPTION: Returns 1 if the command was sent and
| acknowledged or returns 0 if the timeout period for the
| shared memory area elapses.
|
| EXAMPLE:
```

```
|
|          ok = CmSendMuxCommand(
|                   MuxSM,
|                   &C,
|                   MUX_ACTIVATE,
|                   0,
|                   0 );
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Tim Lee from 'SendDataSM'.
-------------------------------------------------------------*/
BOOL
CmSendMuxCommand(
        SM*     M,              // Address of an open shared memory
                                // area read by the Dynamic Manager
                                // of a mux or by one of the input
                                // units (encoders) of the mux.
                                //
        MuxControl* C,          // Address of a mux control record.
                                //
        u8*     OpData,          // Any other input data require by
                                // the command. Use zero if there is
                                // no other data.  THis data must be
                                // in Ictv MPEG Packet format.
                                //
        u32     OpDataSize ) // How many bytes of additional data
                                        // should be appended to the channel
                                        // control record.
{
        u8*                             AtData;
        BOOL                    ok;
        u32             DataSize, PacketSize;
        PacketHeaderSM* H;
        PacketHeaderIM* K;
#ifdef CM_DUMP_TO_FILE
        static u8*              PacketBuffer = 0;
        static u32      PacketBufferSize = 0;

        // If the shared memory space is not yet redirecting
        // output to a file.
        if( ! M->IsSendToFileInstead )
        {
                // Then turn on redirection to a file.
                M->IsSendToFileInstead = 1;
        }

#endif // CM_DUMP_TO_FILE

        // Calculate the overall size of the data portion
        // of the Shared Memory Packet.
        DataSize =
                sizeof( PacketHeaderIM )
                        // For the control packet header.
                +
            sizeof( MuxControl )
                // The size of the control packet data.
            +
            OpDataSize;
                // The size of any additional data tacked
                // on the end.
```

```
        // Calculate the total size of the Shared Memory
        // packet in bytes.
        PacketSize = sizeof( PacketHeaderSM ) + DataSize;

#ifdef CM_DUMP_TO_FILE

        // If the packet size exceeds the current size of
        // the packet buffer.
        if( PacketSize > PacketBufferSize )
        {
                // If there is a current packet buffer.
                if( PacketBuffer )
                {
                        // Free it.
                        free( PacketBuffer );
                }

                // Allocate a new packet buffer.
                PacketBuffer = (u8*) malloc( PacketSize );
                PacketBufferSize = PacketSize;
        }

        // Refer to the first byte of the packet data
        // field.
        AtData = PacketBuffer;

        //
        // Build a mux channel control packet.
        //

        // Refer to the data as an Ictv MPEG packet
        // header.
        K = (PacketHeaderIM*) AtData;

        // Lay down an Ictv MPEG Packet header in
        // the working buffer.
        K->ID = PACKET_ID_MUX_CONTROL;

        // Fill in the packet length field, most
        // significant byte first.
        K->SizeA = (u8) ( sizeof( MuxControl ) >> 16 );
        K->SizeB = (u8) ( sizeof( MuxControl ) >> 8 );
        K->SizeC = (u8) sizeof( MuxControl );

        // Advance to the first byte of the control
        // packet data field.
        AtData += sizeof( PacketHeaderIM );

        // Append the mux control record to the working
        // buffer.
        memcpy( AtData, (u8*) C, sizeof( MuxControl ) );

        // Account for the data added to the working buffer.
        AtData += sizeof( MuxControl );

        // If there is any additional data following the channel
        // control record: the additional data must be
        // Ictv MPEG Packet format.
        if( OpDataSize )
        {
                // Then append the additional data.
                memcpy( AtData, OpData, OpDataSize );
```

```
                // Account for the data added on.
                AtData += OpDataSize;
        }

        // Then send the packet to the output file.
        ok = SendDataSM( M, PacketBuffer, DataSize );

#else // ! CM_DUMP_TO_FILE

        // If the given data size exceeds the size of the
        // target shared memory buffer.
        if( PacketSize > M->BufferSize )
        {
                // Return failure.
                return( 0 );
        }
        // Wait until any data currently in the buffer has been
        // consumed by another party.
        ok = WaitForNoDataSM( M );

        // If there is no data currently in the buffer.
        if( ok )
        {
                // Wait for exclusive access to the shared memory
                // area.
                ok = BeginAccessSM( M );

                // If exclusive access has been acquired before
                // the time out period.
                if( ok )
                {
                        // Refer to the first byte of the shared
                        // memory area as a PacketHeaderSM.
                        H = (PacketHeaderSM*) M->Buffer;

                        // Copy the size to the first four bytes.
                        H->Size = DataSize;

                        // Refer to the first byte of the packet data
                        // field.
                        AtData = M->Buffer + sizeof( PacketHeaderSM );

                        //
                        // Build a mux channel control packet.
                        //

                        // Refer to the data as an Ictv MPEG packet
                        // header.
                        K = (PacketHeaderIM*) AtData;

                        // Lay down an Ictv MPEG Packet header in
                        // the working buffer.
                        K->ID = PACKET_ID_MUX_CONTROL;

                        // Fill in the packet length field, most
                        // significant byte first.
                        K->SizeA = (u8) ( sizeof( MuxControl ) >> 16 );
                        K->SizeB = (u8) ( sizeof( MuxControl ) >> 8 );
                        K->SizeC = (u8) sizeof( MuxControl );

                        // Advance to the first byte of the control
                        // packet data field.
                        AtData += sizeof( PacketHeaderIM );
```

```
                        // Append the mux control record to the working
                        // buffer.
                        memcpy( AtData, (u8*) C, sizeof( MuxControl ) );

                        // Account for the data added to the working buffer.
                        AtData += sizeof( MuxControl );

                        // If there is any additional data following the channel
                        // control record: the additional data must be
                        // Ictv MPEG Packet format.
                        if( OpDataSize )
                        {
                                // Then append the additional data.
                                memcpy( AtData, OpData, OpDataSize );

                                // Account for the data added on.
                                AtData += OpDataSize;
                        }

                        // Mark the presence of the data.
                        HereIsDataSM( M );

                        // Release the shared memory area.
                        EndAccessSM( M );

                        // Wait until the data just put into the buffer
                        // is consumed by another party.
                        ok = WaitForNoDataSM( M );
                }
        }

#endif  // CM_DUMP_TO_FILE

        // Return the status of the send.
        return( ok );
}

/*-------------------------------------------------------------
| NAME: CmSetup
|-------------------------------------------------------------
|
| PURPOSE: To setup the Carousel-to-mux interface.
|
| DESCRIPTION:
|
| NOTE:
|
| HISTORY: [redacted] Tim Lee
|
-------------------------------------------------------------*/
BOOL
CmSetup()
{
        u32     BufferSize;

        // Calculate the size of the Mux shared memory area:
        // always use 128 K for now.
        BufferSize = 128 * 1024;

        // Open the shared memory channel to the mux.
    MuxSM =
     OpenSM(
```

```
        "MuxSM",  // The name of the shared memory area,
                  // a zero-terminated ASCII string up
                  // to 250 bytes long.
                BufferSize );
                                // The size of the shared memory area
                                // in bytes.

        // Allocate a buffer for Ligos output, also 128K.
        CmOutputBuffer = (u8*) malloc( BufferSize );
    CmOutputBufferSize = BufferSize;

        // Return success.
        return( 1 );
}

/*-------------------------------------------------------------
| NAME: CmUpdateChannel
|-------------------------------------------------------------
|
| PURPOSE: To update the channel parameters and/or send
|          a new image bitmap.
|
| DESCRIPTION:
|
| To send a new image through the channel, copy the bitmap
| to the buffer addressed by the 'Buffer' field within
| the 'CmChannelSpec' record.
|
| Use zero for the 'Buffer' field of the channel spec if no
| image data should be sent.
|
| Returns the channel status, S_OK if the update was sucessful
| or E_INVALID_PARAM if not.
|
| EXAMPLE:
|
|   // Declare a channel specification structure.
|       CmChannelSpec C;
|
|   // Call 'CmAddChannel()' here somewhere to make the
|   // channel for "Bis" 207.
|
|   // Then later copy an image bitmap to the input buffer.
|   memcpy( C.Buffer, MyImage, SizeOfMyImage );
|
|   // Then call 'CmUpdateChannel()' to transmit the new image.
|   Result = CmUpdateChannel( &C );
|
| NOTE: TBD: It remains to be determined which channel
|       attributes can be changed after it is created.
|
| ASSUMES:
|
| HISTORY: [redacted] Tim Lee
|          12.15.99 Put in Still Frame packet header.
-------------------------------------------------------------*/
                    // OUT: Result status code:
                    //
                    //   S_OK
                    //   E_INVALID_PARAM
s32
CmUpdateChannel( CmChannelSpec* C )
{
```

```
    CmChannel*              A;
    CmChannelSpec*  D;
    BOOL                    ok;
    u32             FrameSize;

    // Search the channel list for an existing channel with
// the same prefix and virtual channel number.
A = CmFindChannel( C->Prefix, C->Channel );

// If the channel exists.
if( A )
{
    // If the channel parameter specification is valid.
    if( CmIsGoodChannelSpec( C ) )
    {
            // Refer to the currently active channel parameters
            // as 'D'.
            D = &(A->C);

            // If any of the channel parameters are different.
            if( ( D->Prog             != C->Prog    )         ||
                ( D->EncodingBitRate  != C->EncodingBitRate ) ||
                ( D->ChannelBitRate   != C->ChannelBitRate )  ||
                ( D->Hz               != C->Hz      )         ||
                ( D->PID              != C->PID     )         ||
                ( D->PMT              != C->PMT     ) )
            {
                        // Send an update packet to the channel.
                        ok = CmSendChannelCommand( A, CHANNEL_UPDATE, 0, 0
);

                        // Update the current channel parameters.
                        D->Prog             = C->Prog;
                        D->EncodingBitRate  = C->EncodingBitRate;
                        D->ChannelBitRate   = C->ChannelBitRate;
                        D->Hz               = C->Hz;
                        D->PID              = C->PID;
                        D->PMT              = C->PMT;
            }
            else // No channel parmeters changed.
            {
                    // Treat the change status as OK.
                    ok = 1;
            }

            // If there is an image to send.
            if( C->Buffer )
            {

                    // Call the Ligos encoder to encode the image:
                    // the returned data is already in Ictv MPEG
                    // Packet format, specifically a Still Frame Packet.
                    FrameSize =
                            Encode_Frame(
                                    C->Buffer,
                                    CmOutputBuffer );

                    // Fix this.  Causes ligos to flush the previous frame.
                            FrameSize =
                            Encode_Frame(
                                    C->Buffer,
                                    CmOutputBuffer );
```

```
                                // Send the Still Picture packet to the channel with
a
                                // channel prefix packet.
                                //
                                // TBD: Later on grab the shared memory buffer
                                // and use that area as the output of the
                                // Ligos encoder to reduce memory movement.
                                ok &= CmSendChannelCommand(
                                                A,
                                                CHANNEL_DATA,
                                                CmOutputBuffer,
                                                FrameSize );
                        }

                // If all executed commands were acknowledged.
                if( ok )
                {
                        // Return OK status.
                                return( S_OK );
                }
                else // Not acknowledged.
                {
                        // Return failure.
                        return( E_INVALID_PARAM );
                }
        }
        else // Bad channel spec.
        {
                        // Return failure.
                        return( E_INVALID_PARAM );
        }
    }
    else // Channel doesn't exist.
    {
                // Return failure.
                return( E_INVALID_PARAM );
        }
}

/*-------------------------------------------------------------
| NAME: IsASCIILetter
|-------------------------------------------------------------
|
| PURPOSE: To test if a character value is an ASCII letter.
|
| DESCRIPTION: Returns 1 if the character is a letter or 0
| if it isn't.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
|          12.15.99 From 'TLAscii.c'.
-------------------------------------------------------------*/
BOOL
IsASCIILetter( u32 C )
{
        // If the given character is either an lower or upper
        // case letter.
        if( ( C >= 'a' && C <= 'z') ||
```

```
        ( C >= 'A' && C <= 'Z')  )
        {
                return(1);
        }
        else // Not a letter.
        {
                return( 0 );
        }
}
```

ChannelMngr.cpp

```
// ChannelMngr.cpp: implementation of the CChannelMngr class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "carouselexplorer.h"
#include "ChannelMngr.h"
#include "CChannelMap.h"

#include "Channel.h"
#include "StatusFrm.h"
#include "ChannelMapFrm.h"
#include "ScreenImage.h"

#include "..\AlertManager\AlertManagerExports.h"

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

#define QUEUE_THREAD_TIMEOUT    10000

//////////////////////////////////////////////////////////////////////////////
// State Machine

#define SM_CLASS        CChannelMngr

BEGIN_SM_MAP(CChannelMngr)
/*                                                      ST_INACTIVE
                ST_ACTIVE
ST_DEACTIVATING
/*
-------------------------------------------------------------------------------
-----------------
/* SE_ACTIVATE                  */ DO(ST_ACTIVE, OnActivate),         NO_OP,
                                        NO_OP,
/* SE_DEACTIVATE                */ NO_OP,
        DO(ST_DEACTIVATING, OnDeactivate),      DO_NS(OnDeactivate),

/* SE_DEACTIVATED               */ NO_OP,
        NO_OP,
DO(ST_INACTIVE, OnDeactivated),
/* SE_CHANNEL_CREATED     */ NO_OP,
DO_NS(OnChannelCreated),                        DO_NS(OnRemoveChannel),
/* SE_CHANNEL_DESTROYED   */ NO_OP,
DO_NS(OnChannelDestroyed),                      DO_NS(OnRemoveChannel),
/* SE_ENCODE_CHANNEL      */ NO_OP,
DO_NS(OnEncodeChannel),                         NO_OP,
/* SE_ADMIN_CMD                 */ DO_NS(OnCancelAdminCommand),
DO_NS(OnAdminCommand),                          DO_NS(OnCancelAdminCommand),
END_SM_MAP();

BEGIN_IPCTOSM_MAP(CChannelMngr)
END_IPCTOSM_MAP();

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CChannelMngr::CChannelMngr()
: m_CaptureQueueSemaphore(0, 0xFF)
```

```
{
        m_hCaptureThread = NULL;
        m_hHotFolderThread = NULL;
        m_bQuit = FALSE;
        m_nDeactivateCounter = 0;
}

CChannelMngr::~CChannelMngr()
{
}

/*****************************************************************************
*
*       S t a t e   M a c h i n e   V i r t u a l s
*
*****************************************************************************/

void CChannelMngr::OnPreCreate(IPCSM_CREATE_STRUCT* cs)
{
        cs->dwQueueSize = 0xffff;
        cs->dwDataSize = 0;
}

int CChannelMngr::OnInitInstance()
{
        return IQE_SUCCESS;
}

/*****************************************************************************
*
*       S t a t e   M a c h i n e   A c t i o n s
*
*****************************************************************************/

void CChannelMngr::OnActivate(LPARAM lParam1, LPARAM lParam2)
{
        theApp.Log(ARCTOS_EVENT, ARCTOS_CE_ACTIVATE);

        m_bQuit = FALSE;

        // Load the channel map
        if (!LoadChannelMap())
        {
                FireEvent(SE_DEACTIVATE);
                return;
        }

        // Create the capture thread
        DWORD dwThreadId;
        m_CaptureQueueHeartbeat = GetTickCount();
        m_hCaptureThread = CreateThread(NULL, 0, CaptureQueueThreadProc, this, 0,
&dwThreadId);
        ASSERT(m_hCaptureThread);

        // Create the hotfolder thread
        m_hHotFolderThread = CreateThread(NULL, 0, HotFolderThreadProc, this, 0,
&dwThreadId);
        ASSERT(m_hHotFolderThread);

        // Start the watchdog thread
        m_hWatchdogThread = CreateThread(NULL, 0, WatchdogThreadProc, this, 0,
&dwThreadId);
        ASSERT(m_hWatchdogThread);
```

```
}

void CChannelMngr::OnDeactivate(LPARAM lParam1, LPARAM lParam2)
{
        m_bQuit = TRUE;

        // Close the hot folder thread
        if (m_hHotFolderThread)
        {
                // The user has quit - clean up
                if (WAIT_TIMEOUT == WaitForSingleObject(m_hHotFolderThread, 5000))
                {
                        TerminateThread(m_hCaptureThread, FALSE);
                        theApp.Log(ARCTOS_EVENT_WARNING,
ARCTOS_CE_ERR_TERMINATE_THREAD, 1, "hot folder");
                }
                CloseHandle(m_hHotFolderThread);
                m_hHotFolderThread = NULL;
        }

        // Close the watchdog thread thread
        if (m_hWatchdogThread)
        {
                // The user has quit - clean up
                if (WAIT_TIMEOUT == WaitForSingleObject(m_hWatchdogThread, 5000))
                {
                        TerminateThread(m_hWatchdogThread, FALSE);
                        theApp.Log(ARCTOS_EVENT_WARNING,
ARCTOS_CE_ERR_TERMINATE_THREAD, 1, "watchdog");
                }
                CloseHandle(m_hWatchdogThread);
                m_hWatchdogThread = NULL;
        }

        // Close the update thread
        if (m_hCaptureThread)
        {
                // The user has quit - clean up
                if (WAIT_TIMEOUT == WaitForSingleObject(m_hCaptureThread, 5000))
                {
                        TerminateThread(m_hCaptureThread, FALSE);
                        theApp.Log(ARCTOS_EVENT_WARNING,
ARCTOS_CE_ERR_TERMINATE_THREAD, 1, "capture");
                }
                CloseHandle(m_hCaptureThread);
                m_hCaptureThread = NULL;

                // Clean up the capture queue and reset the semaphore
                CSingleLock lock(&m_CaptureQueueMutex, TRUE);
                while (WaitForSingleObject(m_CaptureQueueSemaphore, 0) ==
WAIT_OBJECT_0);
                m_CaptureQueue.RemoveAll();
                lock.Unlock();
        }

        POSITION pos = m_ChannelMap.GetStartPosition();
        if (pos)
        {
                int channelNumber;
                CChannelStub* pChannel;
                while (pos)
                {
                        m_ChannelMap.GetNextAssoc( pos, channelNumber, pChannel );
```

```
                        pChannel->FireEvent(CChannelStub::SE_DESTROY);
                }
        }
        else    // no channels to remove
                FireEvent(SE_DEACTIVATED);
}

void CChannelMngr::OnChannelCreated(LPARAM lParam1, LPARAM lParam2)
{
        // Activate the channel
        CChannelStub* pChannel = (CChannelStub*)lParam1;
        pChannel->FireEvent(SE_ACTIVATE);
}

void CChannelMngr::OnChannelDestroyed(LPARAM lParam1, LPARAM lParam2)
{
        // Look for the channel and remove it
        CChannelStub* pChannel = (CChannelStub*)lParam1;

        CChannelStub* pChannelFound = NULL;
        if (m_ChannelMap.Lookup(pChannel->m_CMR.channel, pChannelFound))
        {
                ASSERT(pChannelFound == pChannel);
                // Remove the channel from the map and delete the channel object
                m_ChannelMap.RemoveKey(pChannelFound->m_CMR.channel);
                delete pChannelFound;
        }
        else
        {
                theApp.Log(ARCTOS_TRACE_1, ARCTOS_CA_ERR_DEACTIVATE, 2,
pChannel->m_CMR.channelName, "Channel not found");
                ASSERT(FALSE);
        }
}

void CChannelMngr::OnRemoveChannel(LPARAM lParam1, LPARAM lParam2)
{
        // Look for the channel and remove it
        CChannelStub* pChannel = (CChannelStub*)lParam1;

        CChannelStub* pChannelFound = NULL;
        if (m_ChannelMap.Lookup(pChannel->m_CMR.channel, pChannelFound))
        {
                ASSERT(pChannelFound == pChannel);
                // Remove the channel from the map and delete the channel object
                m_ChannelMap.RemoveKey(pChannelFound->m_CMR.channel);
                delete pChannelFound;
        }
        else
        {
                theApp.Log(ARCTOS_EVENT_WARNING, ARCTOS_CA_ERR_COMMAND, 2,
pChannel->m_CMR.channelName, "removing", "Channel not found");
        }

        //  If all the channels were removed send a deactivated event
        if (m_ChannelMap.IsEmpty())
                FireEvent(SE_DEACTIVATED);
}

/*
void CChannelMngr::OnChannelDeactivated(LPARAM lParam1, LPARAM lParam2)
{
        // Loop over the list and check if we have anymore active channels
```

```
        CSingleLock lock(&m_ChannelListMutex, TRUE);

//      CChannelStub* pDeactivateChannel = (CChannelStub*)lParam1;

        POSITION pos = m_ChannelList.GetHeadPosition();
        while (pos)
        {
                CChannelStub* pChannel = m_ChannelList.GetNext(pos);
                if (pChannel->GetCurState() != CChannelStub::ST_INACTIVE)
                        return;
        }

        FireEvent(SE_DEACTIVATED);
}
*/
void CChannelMngr::OnDeactivated(LPARAM lParam1, LPARAM lParam2)
{
        theApp.Log(ARCTOS_EVENT, ARCTOS_CE_DEACTIVATE);

        theApp.m_pChannelMapFrm->m_ChannelList.InvalidateRect(NULL, FALSE);
        theApp.m_pStatusFrm->PostMessage(WM_DEACTIVATE_COMPLETE);
}

void CChannelMngr::OnEncodeChannel(LPARAM lParam1, LPARAM lParam2)
{
        CSingleLock lock(&m_CaptureQueueMutex, TRUE);
        m_CaptureQueue.AddTail((CChannelStub*)lParam1);
        m_CaptureQueueSemaphore.Unlock(1);
}

void CChannelMngr::OnAdminCommand(LPARAM lParam1, LPARAM lParam2)
{
        CHANNEL_MAP_ROW*        pCMR = (CHANNEL_MAP_ROW*)lParam1;

        // Find a channelstub object for the channel number (if one exists)
        CChannelStub* pChannel = FindChannel(pCMR->channel);

        // Handle the command
        switch (pCMR->changeFlag)
        {
        case CMD_NONE:
                break;
        //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        // Create a new channel and activate it
        case CMD_ADD:
                {
                        // Does the channel already exist
                        if (pChannel)
                        {
                                theApp.Log(ARCTOS_EVENT_WARNING,
ARCTOS_CA_ERR_COMMAND, 3, pChannel->m_CMR.channelName, "adding", "Channel already
exists");
                                break;
                        }

                        // Create the new channel and activate it
                        pChannel = new CChannelStub(this, pCMR);
                        if (IQE_SUCCESS ==
pChannel->Open(pChannel->m_CMR.channelName, CIpcStateMachine::F_CREATE))
                        {
                                theApp.Log(ARCTOS_TRACE_1, ARCTOS_CA_COMMAND, 2,
pChannel->m_CMR.channelName, "adding");
                                m_ChannelMap.SetAt(pChannel->m_CMR.channel,
```

```
pChannel);
                        }
                        else
                        {
                                theApp.Log(ARCTOS_EVENT_WARNING,
ARCTOS_CA_ERR_COMMAND, 3, pChannel->m_CMR.channelName, "adding", "Can't create IPC
State machine.");
                                delete pChannel;
                        }
                }
                break;
        //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        // Deactivate a channel and remove it
        case CMD_REMOVE:
                if (pChannel == NULL)
                {
                        theApp.Log(ARCTOS_EVENT_WARNING, ARCTOS_CA_ERR_COMMAND, 3,
pChannel->m_CMR.channelName, "removing", "Channel not found.");
                        break;
                }

                pChannel->FireEvent(CChannelStub::SE_DESTROY);

                break;
        //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        // Update channel information
        case CMD_UPDATE:
                if (pChannel)
                {
                        CHANNEL_MAP_ROW* pNewCMR = new CHANNEL_MAP_ROW;
                        memcpy(pNewCMR, pCMR, sizeof(CHANNEL_MAP_ROW));
                        pChannel->FireEvent(CChannelStub::SE_UPDATE,
(LPARAM)pNewCMR);
                }
                else    // Channel not found
                {
                        theApp.Log(ARCTOS_EVENT_WARNING, ARCTOS_CA_ERR_COMMAND, 3,
pChannel->m_CMR.channelName, "updating", "Channel not found.");
                        break;
                }

                break;
        //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        // Get channel status
        case CMD_STATUS:
                break;
        //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        // Activate a channel
        case CMD_ENABLE:
                // Does the channel already exist
                if (pChannel)
                {
                        theApp.Log(ARCTOS_TRACE_1, ARCTOS_CA_COMMAND, 2,
pChannel->m_CMR.channelName, "adding");
                        pChannel->m_CMR.active = TRUE;
                        pChannel->FireEvent(CChannelStub::SE_ACTIVATE);
                }
                else
                {
                        LogInformation(APP_NAME, "Could not find channel to
activate. Channel will be added.");
                        theApp.Log(ARCTOS_EVENT_STRANGE, ARCTOS_CA_ERR_COMMAND, 3,
pChannel->m_CMR.channelName, "enabling", "Channel not found.");
```

```
                        // Create the new channel and activate it
                        pChannel = new CChannelStub(this, pCMR);
                        if (IQE_SUCCESS ==
pChannel->Open(pChannel->m_CMR.channelName, CIpcStateMachine::F_CREATE))
                        {
                                theApp.Log(ARCTOS_TRACE_1, ARCTOS_CA_COMMAND, 2,
pChannel->m_CMR.channelName, "adding");
                                m_ChannelMap.SetAt(pChannel->m_CMR.channel,
pChannel);
                        }
                        else
                        {
                                theApp.Log(ARCTOS_EVENT_WARNING,
ARCTOS_CA_ERR_COMMAND, 3, pChannel->m_CMR.channelName, "adding", "Error creating IPC
State Machine");
                                delete pChannel;
                        }
                        break;
                }
                break;
        //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        // Deactivate a channel and remove it
        case CMD_DISABLE:
                if (pChannel)
                {
                        theApp.Log(ARCTOS_TRACE_1, ARCTOS_CA_COMMAND, 2,
pChannel->m_CMR.channelName, "disabling");
                        pChannel->FireEvent(CChannelStub::SE_DEACTIVATE);
                }
                else
                {
                        theApp.Log(ARCTOS_TRACE_1, ARCTOS_CA_ERR_COMMAND, 3,
pChannel->m_CMR.channelName, "disabling", "Channel not found");
                }
                break;

        default:
                theApp.Log(ARCTOS_EVENT_WARNING, ARCTOS_CA_ERR_COMMAND, 3,
pChannel->m_CMR.channelName, "unknown", "Invalid command");
                break;
        }

        delete pCMR;
}

void CChannelMngr::OnCancelAdminCommand(LPARAM lParam1, LPARAM lParam2)
{
        CHANNEL_MAP_ROW*        pCMR = (CHANNEL_MAP_ROW*)lParam1;
        delete pCMR;
}

/*****************************************************************************
*
*       I n t e r n a l   F u n c t i o n s
*
*****************************************************************************/

BOOL CChannelMngr::LoadChannelMap()
{
        // Get the pathname where the database file is located
        CString strDbFilePath = theApp.m_ArctosSettings.szDatabaseFileName;
        int cPos = strDbFilePath.ReverseFind('\\');
```

```
	if (cPos == -1)
	{
		theApp.Log(ARCTOS_EVENT_DEADLY, ARCTOS_CE_LOAD_CHANNEL_MAP, 2,
theApp.m_ArctosSettings.szDatabaseFileName, "Invalid path");
		return FALSE;
	}
	strDbFilePath = strDbFilePath.Left(cPos);

	// Generate a temp filename for use in the CChannalMap class
	char szTempFile[_MAX_PATH];
	GetTempFileName(strDbFilePath, "map", 0, szTempFile);

	{
		CChannelMap map(theApp.m_ArctosSettings.szDatabaseFileName,
szTempFile);

		// Read the records
		CHANNEL_MAP_ROW cmr;
		if (map.ResetLineRead() == ERROR_SUCCESS)
		{
			theApp.m_pChannelMapFrm->m_ChannelList.SetRedraw(FALSE);

			while (!map.GetNextLine(&cmr))
			{
				if (cmr.channel != 0)
				{
					CChannelStub* pChannel = new
CChannelStub(this, &cmr);
					if (IQE_SUCCESS ==
pChannel->Open(pChannel->m_CMR.channelName, CIpcStateMachine::F_CREATE))
					{
						m_ChannelMap.SetAt(cmr.channel,
pChannel);
					}
					else
					{
						delete pChannel;
						ASSERT(FALSE);
					}
				}
			}

			theApp.m_pChannelMapFrm->m_ChannelList.SetRedraw(TRUE);
		}
		else
		{
			theApp.Log(ARCTOS_EVENT_DEADLY, ARCTOS_CE_LOAD_CHANNEL_MAP,
2, theApp.m_ArctosSettings.szDatabaseFileName, "Error reading database file.");
			return FALSE;
		}
	}

	if (*szTempFile)
		DeleteFile(szTempFile);

	return TRUE;
}

CChannelStub* CChannelMngr::FindChannel(int ChannelNumber)
{
	CChannelStub* pChannel = NULL;
	if (m_ChannelMap.Lookup(ChannelNumber, pChannel))
		return pChannel;
```

```
        else
                return NULL;
}

/*****************************************************************************
*
*       C a p t u r e   Q u e u e   T h r e a d
*
*****************************************************************************/
DWORD CChannelMngr::CaptureQueueThreadProc(LPVOID lpData)
{
        CChannelMngr* pChannelMngr = (CChannelMngr*)lpData;
        return pChannelMngr->CaptureQueueProc();
}

DWORD CChannelMngr::CaptureQueueProc()
{
        CScreenImage Screen(640, 480);
        if (!Screen.Init())
        {
                ASSERT(FALSE);
        }

        while (!m_bQuit)
        {
                m_CaptureQueueHeartbeat = GetTickCount();

                switch (WaitForSingleObject(m_CaptureQueueSemaphore, 100))
                {
                case WAIT_OBJECT_0:
                        {
                                CSingleLock lock(&m_CaptureQueueMutex, TRUE);
                                CChannelStub* pChannelStub =
m_CaptureQueue.RemoveHead();
                                lock.Unlock();

                                // Bring the window to the foreground
                                SetWindowPos(pChannelStub->m_hwndChannelApp,
HWND_TOPMOST,
                                                0,
theApp.m_StatusFrameHeight,
theApp.m_ArctosSettings.ColCount, theApp.m_ArctosSettings.RowCount + 20,
                                                SWP_SHOWWINDOW);

                                Sleep(300);

                                // Wait for the window to redraw
                                for (int i = 0; i < 10 && pChannelStub->IsBusy();
i++, Sleep(100));

                                // Did it redraw or is it taking too long to redraw
                                if (i < 10 || pChannelStub->m_WaitForRedrawCounter >
3)
                                {
                                        if (pChannelStub->m_WaitForRedrawCounter >
3)
                                        {
                                                CString strLog;
                                                strLog.Format("Forced capture of %s
(%s) before redraw",
pChannelStub->m_CMR.channelName, pChannelStub->m_pStatus->szCurrentUrl);
```

```
                        LogInformation(APP_NAME, strLog);
                }

                pChannelStub->m_WaitForRedrawCounter = 0;

                try
                {
                        // Capture the screen image
                        pChannelStub->m_ChannelSpec.Buffer =

(UCHAR*)Screen.GetScreenImage(0,        // src x
                        theApp.m_StatusFrameHeight, // src y
                        theApp.m_ArctosSettings.ColCount, // src width
                        theApp.m_ArctosSettings.RowCount, // src height
                        theApp.m_ArctosSettings.OffsetLeft, // dest x
                        theApp.m_ArctosSettings.OffsetTop); // dest y
                        if
(theApp.m_ArctosSettings.EnableMux)
                        {
pChannelStub->m_ChannelSpec.DataSize = Screen.GetSize();
                                // Send the image to the mux
                                s32 ret = CmUpdateChannel(
&pChannelStub->m_ChannelSpec );
                        }
                }
                catch (...)
                {
                        CString strLog;
                        strLog.Format("Exception calling
CmUpdateChannel for %s",
pChannelStub->m_CMR.channelName);
                        LogInformation(APP_NAME, strLog);
                }

                if (++pChannelStub->m_nNavCounter >
theApp.m_ArctosSettings.NavsBeforeReset)
                {
pChannelStub->FireEvent(CChannelStub::SE_RESET);
                }
                else
                {
pChannelStub->FireEvent(CChannelStub::SE_ENCODED);
                }
        }
        // Send it to the back of the queue
        else
        {
                lock.Lock();
                pChannelStub->m_WaitForRedrawCounter++;
                m_CaptureQueue.AddTail(pChannelStub);
                lock.Unlock();
```

```
                                LogInformation(APP_NAME, "Wating too long
for redraw");
                        }
                }
                break;

        case WAIT_TIMEOUT:
                break;

        default:
                ASSERT(FALSE);
        }
    }

    return 0;
}

/*****************************************************************************
*
*       H o t F o l d e r T h r e a d
*
*****************************************************************************/

DWORD CChannelMngr::HotFolderThreadProc(LPVOID lpData)
{
    CChannelMngr* pChannelMngr = (CChannelMngr*)lpData;
    return pChannelMngr->HotFolderProc();
}

DWORD CChannelMngr::HotFolderProc()
{
    CString strDbFilePath = theApp.m_ArctosSettings.szDatabaseFileName;
    int cPos = strDbFilePath.ReverseFind('\\');
    if (cPos == -1)
    {
        LogInformation(APP_NAME, "Invalid db file path in hot folder proc");
        return 0;
    }

    strDbFilePath = strDbFilePath.Left(cPos);

    // Ask for notification if a file in the admin folder changed
    HANDLE hWait = FindFirstChangeNotification(strDbFilePath, FALSE,
FILE_NOTIFY_CHANGE_LAST_WRITE);

    DWORD dwRet;
    while (!m_bQuit)
    {
        dwRet = WaitForSingleObject(hWait, 100);
        switch (dwRet)
        {
        case WAIT_OBJECT_0:
            {
                // Cancel the change notification so we don't get
bogus changes later
                FindCloseChangeNotification(hWait);

                char szTempFile[_MAX_PATH];
                GetTempFileName(strDbFilePath, "map", 0,
szTempFile);

                {
```

```
map(theApp.m_ArctosSettings.szDatabaseFileName, szTempFile);

                                        // Read the records
                                        if (map.ResetLineRead() == ERROR_SUCCESS)
                                        {
                                                CHANNEL_MAP_ROW* pCMR = new
CHANNEL_MAP_ROW;
                                                while (!map.GetNextChange(pCMR))
                                                {
                                                        FireEvent(SE_ADMIN_CMD,
(LPARAM)pCMR, sizeof(CHANNEL_MAP_ROW));
                                                        pCMR = new CHANNEL_MAP_ROW;
                                                }
                                                delete pCMR;
                                        }
                                        else
                                        {
                                                LogInformation(APP_NAME, "Error
reading database file");
                                        }
                                }

                                DeleteFile (szTempFile);
/*                              CCarouselFile file;
                                _CHANNEL_INFO*  pCi = new _CHANNEL_INFO;

                                CSingleLock lock(&m_ChannelListMutex, TRUE);

                                // Open the database file and look for the next
change
                                if
(file.Open(theApp.m_ArctosSettings.szDatabaseFileName))
                                {

                                        // Loop over changes
                                        while (file.GetNextChangedRecord(pCi))
                                        {
                                                FireEvent(SE_ADMIN_CMD, (LPARAM)pCi,
sizeof(_CHANNEL_INFO));
                                                pCi = new _CHANNEL_INFO;
                                        }

                                        delete pCi;

                                        // Get the next change notification
//                                      file.Flush();
                                        file.Close();
//                                      hWait =
FindFirstChangeNotification(strDbFilePath, FALSE, FILE_NOTIFY_CHANGE_LAST_WRITE);
                                }
*/
                        }
                        hWait = FindFirstChangeNotification(strDbFilePath, FALSE,
FILE_NOTIFY_CHANGE_LAST_WRITE);
                        break;
                case WAIT_TIMEOUT:
                        break;
                default:
                        m_bQuit = TRUE;
                        break;
                }
        }
```

```
        FindCloseChangeNotification(hWait);

        return 0;
}

/*****************************************************************************
*
*       W a t c h d o g T h r e a d
*
*****************************************************************************/

DWORD CChannelMngr::WatchdogThreadProc(LPVOID lpData)
{
        CChannelMngr* pChannelMngr = (CChannelMngr*)lpData;
        return pChannelMngr->WatchdogProc();
}

DWORD CChannelMngr::WatchdogProc()
{
        while (!m_bQuit)
        {
                Sleep(250);
                DWORD curtime = GetTickCount();
                if (curtime - m_CaptureQueueHeartbeat > QUEUE_THREAD_TIMEOUT &&
theApp.m_ArctosSettings.EnableMux)
                {
                        LogInformation(APP_NAME , "Capture queue hung.  System will
shut down");
                        FireEvent(SE_DEACTIVATE);
                }
        }

        return 0;
}
```

TcpEnc.cpp

```
/*-------------------------------------------------------------
| NAME: TcpEnc.cpp
|-------------------------------------------------------------
|
| PURPOSE: To recieve a video PES stream from a TCP port as
|          input to the multiplexer.
|
| DESCRIPTION:
|
|
| HISTORY: [redacted] Tim Lee from SimEnc.cpp by Allan Moluf.
|
| Copyright [redacted] by ICTV, Inc.  All rights reserved.
-------------------------------------------------------------*/

#include <math.h>
#include <stdio.h>
#include "IctvMpeg.h" // ICTV-specific MPEG conventions.
#include "TcpEnc.h"
#include "TsPid.h"
#include "TrialMgr.h"

// CLASS variable declarations

// Register the sub-class TcpEnc for setup-file parsing when
// static objects are initialized at program start up.
Encoder::EncParseObject TcpEnc::s_parseObject( "TCP", "TcpEnc", setup );

#define BYTES_PER_TS_PACKET     188
        // The size of a TS packet.

// Number of frames delay between wait queue exit and mux exit.
#define NORMAL_OUTPUT_DTS_STILL (2.5)
#define NORMAL_OUTPUT_DTS       (3.0)

// MPEG Picture Header Structure
enum
{
        PICTURE_CODING_TYPE_INDEX                = 5, // Offset from first
                                                  // start code byte.
        PICTURE_CODING_TYPE_SHIFT_COUNT = 3,
        PICTURE_CODING_TYPE_MASK                 = 7,
        PICTURE_CODING_TYPE_I                    = 1,
        PICTURE_CODING_TYPE_P           = 2,
        PICTURE_CODING_TYPE_B           = 3
};

// MPEG Start Code Values: all prefixed by three bytes 0, 0, 1.
enum
{
        PICTURE_START_CODE          = 0,
        FIRST_SLICE_START_CODE      = 1,
        LAST_SLICE_START_CODE       = 0xAF,
        RESERVED_START_CODE_A       = 0xB0,
        RESERVED_START_CODE_B       = 0xB1,
        USER_DATA_START_CODE        = 0xB2,
        SEQUENCE_HEADER_START_CODE  = 0xB3,
        SEQUENCE_ERROR_CODE         = 0xB4,
        EXTENSION_START_CODE        = 0xB5,
        RESERVED_START_CODE_C       = 0xB6,
        SEQUENCE_END_CODE           = 0xB7,
        GROUP_START_CODE            = 0xB8,
        FIRST_SYSTEM_START_CODE     = 0xB9,
```

```
        LAST_SYSTEM_START_CODE      = 0xff
};

// Transport Packet Header structure
enum
{
        SYNC_BYTE_INDEX     = 0,
        SYNC_BYTE_VALUE     =   0x47,
        PAYLOAD_START_INDEX     = 1,
        PAYLOAD_START_BIT       =       0x40,
        PID_HI_BYTE_INDEX   = 1,
        PID_HI_BYTE_MASK    =   0x1f,
        PID_LO_BYTE_INDEX   = 2,
        ADAPT_CTRL_INDEX        = 3,
        HAS_ADAPT_BIT           =       0x20,
        HAS_PAYLOAD_BIT         =       0x10,
        CONT_CTR_INDEX          = 3,
        CONT_CTR_MASK           =       0x0f,
        ADAPT_LEN_INDEX         = 4,
        ADAPT_FLAGS_INDEX       = 5,
        ADAPT_DISC_FLAG_BIT     =       0x80,
        ADAPT_PCR_FLAG_BIT      =       0x10,
        ADAPT_PCR_INDEX         = 6
};

// PES Packet Header structure
enum
{
        PACKET_START_INDEX  = 0, // the first 3 bytes should be 00, 00 ,01
        STREAM_ID_INDEX         = 3,
        PES_PACKET_LENGTH_HIBYTE = 4,
        PES_PACKET_LENGTH_LOBYTE = 5,
        DATA_ALIGN_INDEX        = 6,
        DATA_ALIGN_BIT          =       0x04,
        PTS_FLAG_INDEX          = 7,
        PTS_FLAG_BIT            =       0x80,
        DTS_FLAG_INDEX          = 7,
        DTS_FLAG_BIT            =       0x40,
        PES_HDR_LEN_INDEX       = 8,
        PTS_INDEX                       = 9,
        DTS_INDEX                       = 14
};

// This is the data for a dummy coded P-frame.
// 480 scanlines / 16 lines per macro block = 30 slices.
UInt8
DummyCodedPFrame[] =
{
        0, 0, 1, 0x00, 0x0c, 0x90, 0x75, 0x33, 0x80, // PIC HDR
        0, 0, 1, 0xb5, 0x82, 0x2f, 0xf3, 0xc9, 0x80, // PIC COD EXT
        0, 0, 1, 0x01, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 01
        0, 0, 1, 0x02, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 02
        0, 0, 1, 0x03, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 03
        0, 0, 1, 0x04, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 04
        0, 0, 1, 0x05, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 05
        0, 0, 1, 0x06, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 06
        0, 0, 1, 0x07, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 07
        0, 0, 1, 0x08, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 08
        0, 0, 1, 0x09, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 09
        0, 0, 1, 0x0a, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 10
        0, 0, 1, 0x0b, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 11
        0, 0, 1, 0x0c, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 12
        0, 0, 1, 0x0d, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 13
```

```
        0, 0, 1, 0x0e, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 14
        0, 0, 1, 0x0f, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 15
        0, 0, 1, 0x10, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 16
        0, 0, 1, 0x11, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 17
        0, 0, 1, 0x12, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 18
        0, 0, 1, 0x13, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 19
        0, 0, 1, 0x14, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 20
        0, 0, 1, 0x15, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 21
        0, 0, 1, 0x16, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 22
        0, 0, 1, 0x17, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 23
        0, 0, 1, 0x18, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 24
        0, 0, 1, 0x19, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 25
        0, 0, 1, 0x1a, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 26
        0, 0, 1, 0x1b, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 27
        0, 0, 1, 0x1c, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 28
        0, 0, 1, 0x1d, 0x2a, 0x70, 0x10, 0x33, 0x80, // SLICE 29
        0, 0, 1, 0x1e, 0x2a, 0x70, 0x10, 0x33, 0x80  // SLICE 30
};

TcpLogRecord    TheLogRecord;
                                // The current log record.

#define LOG_RECORD_COUNT                1000
                // How many log records should be collected, one
                // record per output frame.

extern "C"
{

u32             GetQuantiserScaleCode( u8* );
void    IncrementTemporalSequence( u8* );
void    PutQuantiserScaleCode( u8*, u32 );
void    SetPictureCodingType( u8*, u32 );
u8*             ToSlice( u8*, u8* );
u8*             ToSpecificStartCode( u8*, u8*, u32 );

}

/*-------------------------------------------------------------
| NAME: GetQuantiserScaleCode
|-------------------------------------------------------------
|
| PURPOSE: To get the value from the quantiser_scale_code
|          field of a picture slice.
|
| DESCRIPTION: Given the address of first byte of a picture
| slice this routine fetches the five bit value held in the
| quantiser_scale_code field.
|
| EXAMPLE:
|
|       qsc = GetQuantiserScaleCode( AtSlice );
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
u32
GetQuantiserScaleCode( u8* AtSlice )
{
        u32 n;
```

```
        // The quantiser_scale_code field is 5 bits long and it
        // starts at byte offset 4 from the beginning of the
        // slice, being held in the high five bits of the byte.

        // Fetch the value and shift it into the low bits of
        // a 32-bit value.
        n = (u32) ( ( AtSlice[4] >> 3 ) & 0x1F );

        // Return the result.
        return( n );
}

/*-------------------------------------------------------------
| NAME: IncrementTemporalSequence
|-------------------------------------------------------------
|
| PURPOSE: To increment the value held in the temporal
|          reference field of a Picture Header
|
| DESCRIPTION: Given the address of a Picture Header this
| routine increments the value in the temporal reference
| field.
|
| EXAMPLE:
|
|        IncrementTemporalSequence( AtPicture );
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
void
IncrementTemporalSequence( u8* AtPicture )
{
        u32 TempRef;


        // The temporal reference field is 10 bits long and it
        // starts at byte offset 4 from the beginning of the
        // Picture Header.

        // Get the high 8 bits.
        TempRef = AtPicture[4] << 2;

        // Merge the low 2 bits.
        TempRef |= ( AtPicture[5] >> 6 ) & 3;

        // Increment the temporal reference number.
        TempRef++;

        // Put back the high 8 bits.
        AtPicture[4] = (u8) ( TempRef >> 2 );

        // Put back the low 2 bits.
        AtPicture[5] = ( AtPicture[5] & 0x3f ) |
                       ( ( TempRef & 3 ) << 6 );
}

/*-------------------------------------------------------------
| NAME: MakeDummyCodedPFrame
```

```
/*-------------------------------------------------------------

PURPOSE: To make a dummy coded P picture that repeats a
         given I picture.

DESCRIPTION: Returns a P picture at the destination buffer
and the return value is the size of the P picture in
bytes.

Returns zero if no valid P picture could be constructed.

EXAMPLE:

ByteCount = MakeDummyCodedPFrame(
                AtIPicture,
                AtIPictureEnd,
                AtPPicture );

NOTE:

ASSUMES:

HISTORY: [redacted]
-------------------------------------------------------------*/
                    // Returns the number of bytes in the
u32                 // resulting P Picture.
MakeDummyCodedPFrame(
        u8* AtIPicture, // The first byte of the Picture Header
                        // of the I Picture used as the source.
                        //
        u8* AtIPictureEnd,
                        // The first byte after the last byte in
                        // the I Picture used as the source.
                        //
        u8* AtPPicture )// The destination buffer where the
                        // P Picture will be stored.
{
        u8* AtPPictureEnd;
        u8* AtSliceI;
        u8* AtSliceP;
        u32 Q, ByteCount;
        u32 PictureHeaderByteCount;
        u32 SliceNumber;

        // If the source picture is missing.
        if( ! AtIPicture )
        {
                // Return zero to signal an error.
                return( 0 );
        }

        // Find the first slice in the source picture.
        AtSliceI = ToSlice( AtIPicture, AtIPictureEnd );

        // If no slice was found in the picture.
        if( ! AtSliceI )
        {
                // Return zero to signal an error.
                return( 0 );
        }

        // Calculate the number of bytes in the picture
        // header plus picture code extension.
```

```
PictureHeaderByteCount = AtSliceI - AtIPicture;

// Copy the picture header to the destination buffer.
memcpy( AtPPicture,
        AtIPicture,
        PictureHeaderByteCount );

// Account for the new header bytes added to the
// P Picture.
AtPPictureEnd = AtPPicture + PictureHeaderByteCount;

// Increment the value of the temporal sequence
// number in the new picture header.
IncrementTemporalSequence( AtPPicture );

// Set the picture coding type to P type.
SetPictureCodingType( AtPPicture, (u32) PICTURE_CODING_TYPE_P );

// For every slice in the source picture.
while( AtSliceI )
{
        // Get the slice number from the current slice.
        SliceNumber = (u32) AtSliceI[3];

        // Find the corresponding slice in the template dummy
        // coded P picture.
        AtSliceP =
                ToSpecificStartCode(
                        DummyCodedPFrame,
                                // Address of the byte where the search
                        // begins.
                        //
                        DummyCodedPFrame + sizeof( DummyCodedPFrame ),
                                // Address of the first byte after
                                // the last valid byte in the range.
                        //
                        SliceNumber );
                                // The value of the fourth byte of
                                // the target Start Code.

        // If the P slice was found.
        if( AtSliceP )
        {
                // Copy the slice template data to the P picture
                // buffer: there are nine bytes in each dummy
                // slice.
                memcpy( AtPPictureEnd,
                        AtSliceP,
                        (u32) 9 );

                // Account for the new header bytes added to the
                // P Picture.
                AtPPictureEnd += 9;

                // Get the quantiser_scale_code from the
                // I picture slice.
                Q = GetQuantiserScaleCode( AtSliceI );

                // Put the quantiser_scale_code into the P
                // slice.
                PutQuantiserScaleCode( AtSliceP, Q );
        }
```

```
                // Advance to the next I slice.
                AtSliceI = ToSlice( AtSliceI + 4, AtIPictureEnd );
        }

        // Calculate the number of bytes in the P picture.
        ByteCount = AtPPictureEnd - AtPPicture;

        // Return the picture size.
        return( ByteCount );
}

/*-------------------------------------------------------------
| NAME: PutQuantiserScaleCode
|-------------------------------------------------------------
|
| PURPOSE: To put a value into the quantiser_scale_code
|          field of a picture slice.
|
| DESCRIPTION: Given the address of first byte of a picture
| slice this routine stores a five bit value into the
| quantiser_scale_code field.
|
| EXAMPLE:
|
|        PutQuantiserScaleCode( AtSlice, NewValue );
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
void
PutQuantiserScaleCode( u8* AtSlice, u32 NewValue )
{
        u8      A;

        // The quantiser_scale_code field is 5 bits long and it
        // starts at byte offset 4 from the beginning of the
        // slice, being held in the high five bits of the byte.

        // Fetch the byte holding the field.
        A = AtSlice[4];

        // Punch a hole for the new value: the low 3 bits are
        // preserved.
        A &= 7;

        // Shift and merge the new value.
        A |= (u8) ( NewValue << 3 );

        // Put the whole byte back.
        AtSlice[4] = A;
}

/*-------------------------------------------------------------
| NAME: SetPictureCodingType
|-------------------------------------------------------------
|
| PURPOSE: To set value of the Picture Coding Type field in
|          a Picture Header.
|
| DESCRIPTION: Given the address of a Picture Header this
```

```
| routine stores a value into the Picture Coding Type field.
|
| EXAMPLE:
|
|  SetPictureCodingType( AtPicture, PICTURE_CODING_TYPE_P );
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
void
SetPictureCodingType(
        u8* AtPicture,          // Address of a Picture Header.
                                //
        u32 PictureCodingType ) // Must be one of these values:
                                //   PICTURE_CODING_TYPE_I
                                //   PICTURE_CODING_TYPE_P
                                //   PICTURE_CODING_TYPE_B
{
        u8 A;

        // Fetch the byte containing the field.
        A = AtPicture[ PICTURE_CODING_TYPE_INDEX ];

        // Punch a hole for the new value.
        A = A &
            ( ~( (u8) PICTURE_CODING_TYPE_MASK <<
                 PICTURE_CODING_TYPE_SHIFT_COUNT ) );

        // Merge the new value into the byte.
        A |= (u8) ( PictureCodingType <<
                    PICTURE_CODING_TYPE_SHIFT_COUNT );

        // Store the byte back to the picture header.
        AtPicture[ PICTURE_CODING_TYPE_INDEX ] = A;
}

/*-------------------------------------------------------------
| NAME: TcpEnc::TcpEnc
|-------------------------------------------------------------
|
| PURPOSE: To make a TcpEnc object.
|
| DESCRIPTION:
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
|          [redacted] Added PES arrival time inits.
-------------------------------------------------------------*/
TcpEnc::TcpEnc() : Encoder("TcpEnc"),
        d_isVideo(false),
        d_InputBufSize( 1024*128 ),
        d_InputFrameCount(0),
        d_OutputFrameCount(0),
        d_totalPesData(0.0),
        d_dbgLev(0),
```

```
        d_BytesReceived(0),
        d_IsConnected( 0 ),
        d_LogBuffer( 0 ),
        d_LogBufferEnd( 0 ),
        d_IsFirstFrame( 1 )
{
        // use ctor-initializers where possible
        // then allocate memory and initialize other members

        // Allocate the input buffer to hold data read from the
        // TCP socket.
        d_InputBuf = (UInt8*) malloc( d_InputBufSize );

        // Set the timestamp base offset to zero.
        d_TimeStamp.setBaseTime( 0.0 );
}

/*-------------------------------------------------------------
| NAME: TcpEnc::~TcpEnc
|-------------------------------------------------------------
|
| PURPOSE: To delete a TcpEnc object.
|
| DESCRIPTION:
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
TcpEnc::~TcpEnc()
{
        // Delete any owned objects and allocated space.

        // Close any open socket connection and discard the
        // socket.
        CloseConnection();

        // If an input buffer is allocated.
        if( d_InputBuf )
        {
                free( d_InputBuf );
        }
}

/*-------------------------------------------------------------
| NAME: TcpEnc::AdjustFrameTimeStamps
|-------------------------------------------------------------
|
| PURPOSE: To adjust the any PTS/DTS values in the current
|          PES frame.
|
| DESCRIPTION:
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES: STC time refers to the byte currently leaving
|          the mux.
```

```
|
| HISTORY: [redacted] Copied from SimEnc::threadStep.
------------------------------------------------------------*/
void
TcpEnc::AdjustFrameTimeStamps()
{
        double  delta;

        // Insert current PTS/DTS.
        // TBD exactly how we want to express this -- the basic
        // idea is that for simulation (or most real encoding) we
        // expect to get a steady increase in DTS values
        // (when PTS != DTS, the PTS will follow ths DTS by some
        // number of frame periods).  We have modelled this in
        // outputDts(), which gets initialized by the offset we
        // want at the first PES buffer with a PTS.
        //
        // Now if PTS==DTS, we expect them to be the next value
        // (using dtsInc()).
        //
        // If PTS and DTS are both specified, then DTS follows
        // this simple pattern and we make the adjusted PTS keep
        // the same offset.
        if( pes()->ptsFlag() )
        {
                // If there is a DTS field in the PES header.
                if( pes()->dtsFlag() )
                {
                        // Calculate how much delay is currently
                        // specified between the PTS and the DTS.
                        delta = pes()->pts() - pes()->dts();

                        // Set the DTS to the value set when the PES
                        // buffer is output.
                        pes()->changeDts( outputDts() );

                        // Maintain the original delay between the DTS
                        // and the the PTS.
                        pes()->changePts( outputDts() + delta );
                }
                else // There is no DTS field.
                {
                        // Set the PTS to the value set when the PES
                        // buffer is output.
                        pes()->changePts( outputDts() );
                }
        }

        // If debug info should be output.
        if( d_dbgLev & 2 )
        {
                if( pes()->ptsFlag() )
                {
                        cout << TimeStamp::ts() << " "
                        << threadName() << " threadStep  newPts="
                        << Util::formatFixed(pes()->pts(),5);

                        if( pes()->ptsFlag() )
                        {
                                cout << " newDts=" <<
Util::formatFixed(pes()->dts(),5);
                        }
```

```
                        cout << " outputDts=" << Util::formatFixed(outputDts(),5) <<
endl;
                }
        }
}

/*-------------------------------------------------------------
| NAME: TcpEnc::CloseConnection
|-------------------------------------------------------------
|
| PURPOSE: To close any open TCP connection.
|
| DESCRIPTION: If the socket is already closed then nothing
| is done.
|
| If connected then:
|
| 1. If the socket exists to be closed it is closed.
|
| 2. The connection state is set to not connected.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Factored out of TcpEnc::threadStep().
-------------------------------------------------------------*/
void
TcpEnc::CloseConnection()
{
        TcpLogRecord*   AtLogRecord;

        // If the TCP socket is connected.
        if( d_IsConnected )
        {
                // If a valid data socket is open.
                if( d_Socket )
                {
                        // Close the socket.
                        Socket_Close( d_Socket );

                        // Mark the socket as deleted.
                        d_Socket = 0;
                }

                // If a valid listening socket is open.
                if( d_ListenSocket )
                {
                        // Close the socket.
                        Socket_Close( d_ListenSocket );

                        // Mark the socket as deleted.
                        d_ListenSocket = 0;
                }

                // Set the socket state to disconnected.
                d_IsConnected = 0;

                // So there are no bytes that can be read.
                d_BytesAvailable = 0;
```

```
                // If dumping PES packet data to a log file.
                if( d_dbgLev == 10 )
                {
                        // Close the log file.
                        fclose( d_LogFile );

                        // And exit.
                        exit(0);
                }

                // If the debug log should be written.
                if( d_dbgLev == 9 )
                {
                        // Then open a special log file.
                        d_LogFile = fopen( "TcpLog.txt", "w" );

                        // Refer to the first log record.
                        AtLogRecord = (TcpLogRecord*) d_LogBuffer;

                        // For each log record.
                        while( ( (u8*) AtLogRecord ) < d_LogBufferEnd )
                        {
                                // Output the STC followed by outputDts
                                // and the frame number.
                                fprintf( d_LogFile,
                                         "%f\t%f\t%f\t%f\t%d\n",
                                         AtLogRecord->Stc,
                                         AtLogRecord->RequestedWait,
                                         AtLogRecord->ActualWait,
                                         AtLogRecord->OutputDts,
                                         AtLogRecord->InputFrameNumber );

                                // Advance to the next frame.
                                AtLogRecord++;
                        }

                        // Close the log file.
                        fclose( d_LogFile );

                        // Clear the file handle.
                        d_LogFile = 0;

                        // Free the log buffer.
                        free( d_LogBuffer );

                        // Mark the buffer as missing.
                        d_LogBuffer      = 0;
                        d_LogBufferEnd   = 0;
                        d_LogRecordCount = 0;

                        // Quit the application.
                        exit( 0 );
                }
        }
}

/*-------------------------------------------------------------
| NAME: TcpEnc::NormalizeOutputDts
|-------------------------------------------------------------
|
| PURPOSE: To keep the outputDts value within bounds.
|
| DESCRIPTION: OutputDts is the earliest time a frame can
```

```
| leave a frame input queue for input into the mux.
|
| STC is the point in time where a byte is leaving the mux.
|
| OutputDTS is initialized to the current STC adjusted by
| the amount of time it takes a frame to flow through the
| mux.
|
| For frames after the first frame outputDts is advanced by
| one frame time each time a new frame is added to the frame
| input queue using enqueuePesBuf().
|
| Sometimes the outputDts value departs widely from its normal
| value of Stc + DtsOffset.
|
| If the deviation is more than an amount that might cause
| confusion downstream then some sort of correction must be
| made.
|
| OUTPUTDTS NORMALIZATION RULE: The current convention is to
| reset the OutputDts to Stc + DtsOffset if the tolerance
| level is exceeded.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]  Copied from SimEnc::threadStep.
|          [redacted]  Revised to protect against out of bounds
|                      conditions that corrupt data in the
|                      pipeline.
|          [redacted]  Factored out ResetOutputDts.
|                      Cleaned up and redocumented, renamed from
|                      ComputeOutputDts.
-------------------------------------------------------------*/
void
TcpEnc::NormalizeOutputDts()
{
        double Stc, DtsOffset, OutputDts;
        double OutputDtsTol, OutputDtsDev;

        // Get local copies of the values we need.
        Stc       = Util::stcTime();
        DtsOffset = dtsOffset();
        OutputDts = outputDts();

        // Use 1/2 DtsOffset as the tolerance.
        OutputDtsTol = DtsOffset * .5;

        // Compute the absolute deviation from the norm.
        OutputDtsDev = fabs( Stc + DtsOffset - OutputDts );

        // If the deviation exceeds the tolerance.
        if( OutputDtsDev > OutputDtsTol )
        {
                // Reset the OutputDTS to be the current STC
                // adjusted by the amount of time it normally takes
                // a frame to flow through the mux.
                ResetOutputDts();
        }
}
```

```
/*-------------------------------------------------------------
 NAME: TcpEnc::setup
-------------------------------------------------------------

 PURPOSE: To parse the configuration file to set up a TCP
          type input.

 DESCRIPTION: setup() handles these setup parameters:

       Type=xxx[,id]   Required Video/AC3/Audio
                    ( Audio means MPEG-1 audio)
                                    Use Type=Video,0x80 for GI video

       Prog=nnn        Required        Program number in mux output
       Port=xxx        Required        Input TCP port number
       Rate=nnn.n      Required        ES rate in bits/sec
       PID=nnn         Required        ES PID in mux output
       Hz=nnn.n        Optional        24/25/29.97/30/50/60 (for video only)
       PMT=nnn         Optional        PMT PID in mux output

   TimeOut=n.n Optional    The maximum number of seconds the
                           TCP connection may remain open
                           without receiving data before
                           the connection is closed and
                           reset.  A multiplier of 5 is
                           applied to the TimeOut parameter
                           for the period between the initial
                           connection and the arrival of
                           the first byte.  The default value
                           for TimeOut is 2 seconds.

   Dbg=nn              Optional        Debug options

 If all the required parameters are found, all values are
 valid, and the port can be opened, we create a TcpEnc object
 and return a pointer to it.  Otherwise, we output an error
 message and return NULL.

 The parser may also throw an exception for some bad input
 text.

 EXAMPLE:

 NOTE:

 ASSUMES:

 HISTORY: [redacted]
          [redacted] Added TimeOut parameter.
-------------------------------------------------------------*/
Encoder*
TcpEnc::setup( NvParse &parse )
{
        bool                    haveType;
        bool                    haveProg;
        bool                    havePort;
        bool                    haveRate;
        bool                    havePID;
        bool                    haveHz;
        bool                    havePMT;
        bool                    haveTimeOut;
        bool                    haveDbg;
```

```
UInt8                   streamType;
UInt16                  progNum;
TsPkt::PidType  esPid;
TsPkt::PidType  pmtPid;
double                  bitRate;
double                  TimeOut;
double                  dtsInc;
double                  defaultDtsInc;
int                     nVals;
char*                   endPosn;
UInt32                  tVal;
Int32           PortNumber;
const string*   nam;
const string*   val;
TcpEnc*                 enc;
UInt32                  dbgLev;

// Start with no input parameters parsed.
haveType    = false;
haveProg    = false;
havePort    = false;
haveRate    = false;
havePID     = false;
haveHz      = false;
havePMT     = false;
haveTimeOut = false;
haveDbg     = false;

// Until all parameters have been parsed.
while( 1 )
{
        parse.advance();

        nVals = parse.valCount();

        if( nVals < 0 )
        {
                break;
        }

        nam = parse.name();

        // If the parameter name is recognized.
        if( *nam == "Type"    ||
            *nam == "Prog"    ||
            *nam == "Port"    ||
            *nam == "PID"     ||
            *nam == "PMT"     ||
            *nam == "Hz"      ||
            *nam == "Rate"    ||
            *nam == "TimeOut" ||
                *nam == "Dbg" )
        {
                ; // Proceed.
        }
        else // The parameter name is not recognized.
        {
                // Report the error.
                cout << "Parse error at line " << parse.lineNum() <<
                ":  Unexpected parameter " << *nam << endl;

                cerr << "Parse error at line " << parse.lineNum() <<
                ":  Unexpected parameter " << *nam << endl;
```

```
        throw ExcpObj("TcpEnc::parse internal error for " + *nam);

        return NULL;
}

// if any parameters have other than 1 value, handle first
if( *nam == "Type" )
{
        // TBD complain if repeated
        if( nVals < 1 || nVals > 2 )
        {
                // Report the error.
                cout << "Parse error at line " << parse.lineNum() <<
                ":  " << *nam << "= expects 1 or 2 values, not " <<
                nVals << endl;

                cerr << "Parse error at line " << parse.lineNum() <<
                ":  " << *nam << "= expects 1 or 2 values, not " <<
                nVals << endl;

                return NULL;
        }

        val = parse.val(0);

        // If the stream type is recognized.
        if( *val == "Video" ||
            *val == "Audio" ||
            *val == "AC3" )
        {
                ; // Proceed.
        }
        else // The stream type is not recognized.
        {
                // Report the error.
                cout << "Parse error at line " <<
                parse.lineNum() << ":  bad value for " <<
                *nam << "=" << *val << endl;

                cerr << "Parse error at line " <<
                parse.lineNum() << ":  bad value for " <<
                *nam << "=" << *val << endl;

                return NULL;
        }

        if( *val == "Video" )
        {
                streamType = TsPkt::MPEG2_VIDEO_STREAM;

                // 29.97 Hz
                defaultDtsInc = 3003.0;
        }

        if( *val == "Audio" )
        {
                streamType = TsPkt::MPEG2_AUDIO_STREAM;

                // 41.667 Hz = 48 KHz / 1152 samples/frame
                defaultDtsInc = 2160.0;
        }
```

```
                if( *val == "AC3" )
                {
                        streamType = TsPkt::AC3_AUDIO_STREAM;

                        // 31.25 Hz = 48 KHz / 1536 samples/frame
                        defaultDtsInc = 2880.0;
                }

                // If there is a second value associated with
                // the 'Type' parameter.
                if( nVals == 2 )
                {
                        val = parse.val(1);

                        tVal = strtoul( val->c_str(), &endPosn, 0 );

                        if( *endPosn != '\0' || tVal > 0xff )
                        {
                                cout << "Parse error at line " <<
                                parse.lineNum() << ":  bad number value for
" <<

                                *nam << "=" << *val << endl;

                                cerr << "Parse error at line " <<
                                parse.lineNum() << ":  bad number value for
" <<

                                *nam << "=" << *val << endl;

                                return NULL;
                        }

                        streamType = tVal;
                }

                haveType = true;

                continue;
        }

        // All the remaining parameters expect 1 value.
        if( nVals != 1 )
        {
                cout << "Parse error at line " << parse.lineNum() <<
                ":  " << *nam << "= expects 1 value, not " <<
                nVals << endl;

                cerr << "Parse error at line " << parse.lineNum() <<
                ":  " << *nam << "= expects 1 value, not " <<
                nVals << endl;

                return NULL;
        }

        val = parse.val(0);

        if( *nam == "Prog" )
        {
                // TBD complain if repeated
                progNum = strtoul(val->c_str(), &endPosn, 0);

                if( *endPosn != '\0' )
                {
                        cout << "Parse error at line " << parse.lineNum() <<
```

```
                ": bad number value for " << *nam << "=" <<
                *val << endl;

                cerr << "Parse error at line " << parse.lineNum() <<
                ": bad number value for " << *nam << "=" <<
                *val << endl;

                return NULL;
        }

        haveProg = true;

        continue;
}

if( *nam == "Port" )
{
        // TBD complain if repeated

        // Convert the port number string to an integer.
        PortNumber = strtoul( val->c_str(), &endPosn, 0 );

        havePort = true;

        continue;
}

if( *nam == "PID" )
{
        // TBD complain if repeated
        esPid = strtoul( val->c_str(), &endPosn, 0 );

        if( *endPosn != '\0' )
        {
                cout << "Parse error at line " << parse.lineNum() <<
                ": bad number value for " << *nam << "=" <<
                *val << endl;

                cerr << "Parse error at line " << parse.lineNum() <<
                ": bad number value for " << *nam << "=" <<
                *val << endl;

                return NULL;
        }

        // Check to see if the PID is already reserved.
        if( TsPid::isPidReserved(esPid) )
        {
                cout << "Parse error at line " << parse.lineNum() <<
                ": PID is already reserved for " << *nam << "=" <<
                *val << endl;

                cerr << "Parse error at line " << parse.lineNum() <<
                ": PID is already reserved for " << *nam << "=" <<
                *val << endl;

                return NULL;
        }

        // Reserve the PID.
        TsPid::reservePid(esPid);

        havePID = true;
```

```
                continue;
        }

        if( *nam == "PMT" )
        {
                // TBD complain if repeated

                pmtPid = strtoul(val->c_str(), &endPosn, 0);

                if( *endPosn != '\0' )
                {
                        cout << "Parse error at line " << parse.lineNum() <<
                        ":  bad number value for " << *nam << "=" <<
                        *val << endl;

                        cerr << "Parse error at line " << parse.lineNum() <<
                        ":  bad number value for " << *nam << "=" <<
                        *val << endl;

                        return NULL;
                }

                havePMT = true;

                continue;
        }

        if( *nam == "Hz" )
        {
                // TBD complain if repeated
                // *val * ptsInc should equal 90000

                dtsInc = 0.;

                if( *val == "23.976" ) dtsInc = 3753.75;
                if( *val == "24" )     dtsInc = 3750.0;
                if( *val == "25" )     dtsInc = 3600;
if( *val == "29.97" )  dtsInc = 3003.0;
if( *val == "30" )     dtsInc = 3000.0;
if( *val == "50" )     dtsInc = 1800.0;
                if( *val == "59.94" )  dtsInc = 1501.5;
if( *val == "60" )     dtsInc = 1500.0;

// If the frame rate is not valid.
if( dtsInc == 0. )
{
                        // Report the error.
                        cout << "Parse error at line " << parse.lineNum() <<
                        ":  bad number value for " << *nam << "=" <<
                        *val << endl;

                        cerr << "Parse error at line " << parse.lineNum() <<
                        ":  bad number value for " << *nam << "=" <<
                        *val << endl;

                        return NULL;
                }

                haveHz = true;

                continue;
        }
```

```
                if( *nam == "Rate" )
                {
                        // TBD complain if repeated
                        bitRate = strtod(val->c_str(), &endPosn);

                        if( *endPosn != '\0' )
                        {
                                cout << "Parse error at line " << parse.lineNum() <<
                                ":  bad number value for " << *nam << "=" <<
                                *val << endl;

                                cerr << "Parse error at line " << parse.lineNum() <<
                                ":  bad number value for " << *nam << "=" <<
                                *val << endl;

                                return NULL;
                        }

                        haveRate = true;

                        continue;
                }

                if( *nam == "TimeOut" )
                {
                        // TBD complain if repeated
                        TimeOut = strtod(val->c_str(), &endPosn);

                        if( *endPosn != '\0' )
                        {
                                cout << "Parse error at line " << parse.lineNum() <<
                                ":  bad number value for " << *nam << "=" <<
                                *val << endl;

                                cerr << "Parse error at line " << parse.lineNum() <<
                                ":  bad number value for " << *nam << "=" <<
                                *val << endl;

                                return NULL;
                        }

                        haveTimeOut = true;

                        continue;
                }

                if( *nam == "Dbg" )
                {
                        //  Dbg=nn                          --opt--         0 disables
debug prints
                        // TBD complain if repeated
                        dbgLev = strtoul(val->c_str(), &endPosn, 0);

                        if( *endPosn != '\0' )
                        {
                                cout << "Parse error at line " << parse.lineNum() <<
                                ":  bad number value for " << *nam << "=" << *val <<
endl;
                                cerr << "Parse error at line " << parse.lineNum() <<
                                ":  bad number value for " << *nam << "=" << *val <<
endl;
```

```
                return NULL;
            }

            haveDbg = true;
        }
    } // end while

    // Verify that all of the required parameters have been parsed.
    if( !haveType ||
        !haveProg ||
        !havePort ||
        !haveRate ||
        !havePID )
    {
        cout << "Parse error at line " << parse.lineNum() <<
        ":  missing required parameter(s):" <<
        (haveType ? "" : "  Type=") <<
        (haveProg ? "" : "  Prog=") <<
        (havePort ? "" : "  Port=") <<
        (haveRate ? "" : "  Rate=") <<
        (havePID  ? "" : "  PID=" ) << endl;

        cerr << "Parse error at line " << parse.lineNum() <<
        ":  missing required parameter(s):" <<
        (haveType ? "" : "  Type=") <<
        (haveProg ? "" : "  Prog=") <<
        (havePort ? "" : "  Port=") <<
        (haveRate ? "" : "  Rate=") <<
        (havePID  ? "" : "  PID=" ) << endl;

        return NULL;
    }

    // If the PMT PID parameter has been specified.
    if( havePMT )
    {
        // Verify that the PMT PID can or has been associated
        // with the given program number.
        if( ! TrialMgr::checkPmtForProg( pmtPid, progNum ) )
        {
            cout << "Setup error at line " << parse.lineNum()
            << "  PMT=" << pmtPid
            << " is invalid for program " << progNum << endl;

            cerr << "Setup error at line " << parse.lineNum()
            << "  PMT=" << pmtPid
            << " is invalid for program " << progNum << endl;

            return NULL;
        }
    }

    // At this point all of the required parameters have been
    // validly parsed.

    // Create a new TcpEnc object.
    enc = new TcpEnc();

    // Depending on the stream type.
    switch( streamType )
    {
        case TsPkt::AC3_AUDIO_STREAM:
        {
```

```
                // Mark this input stream as non-video.
                enc->d_isVideo = false;

                // Make a new Ac3Pes object and store it in the
                // 'd_pes' field inherited from 'Encoder'.
                enc->setPes( new Ac3Pes );

                break;
        }

        case TsPkt::MPEG2_AUDIO_STREAM:
        {
                // Mark this input stream as non-video.
                enc->d_isVideo = false;

                // Make a new AudioPes object and store it in the
                // 'd_pes' field inherited from 'Encoder'.
                enc->setPes( new AudioPes );

                break;
        }

        default: // All other types are video.
        {
                // Mark this input stream as being video.
                enc->d_isVideo = true;

                // Make a new VideoPes object and store it in the
                // 'd_pes' field inherited from 'Encoder'.
                enc->setPes( new VideoPes );
        }
}

// Set the mux output PID for the elementary stream.
enc->setEsPid( esPid );

// If the PMT PID has been specified for the program
// that contains the elementary stream.
if( havePMT )
{
        // Save the PMT PID in the 'd_pmtPid' field
        // inherited from 'Encoder'.
        enc->setPmtPid( pmtPid );
}
else // The PMT PID is not specified.
{
        // Then the value of the 'd_pmtPid' field keeps
        // the initial value that it was given when the
        // object was created: TsPkt::TS_NULL_PID,
        // see constructor for 'Encoder'.
        ;
}

// If the video frame rate has been specified.
if( haveHz )
{
        // If this is not a video stream.
        if( ! enc->d_isVideo )
        {
                cout << "Setup error at line " << parse.lineNum() <<
                ":  Hz= allowed only for video " << endl;

                cerr << "Setup error at line " << parse.lineNum() <<
```

```
                        ":  Hz= allowed only for video " << endl;

                        delete enc;

                        return NULL;
                }

                // Convert dtsInc units of 90 KHz to seconds.
                enc->setDtsInc( dtsInc / 90000.0 );
        }
        else // The video frame rate has not been specified.
        {
                // Convert defaultDtsInc units of 90 KHz to seconds.
                enc->setDtsInc( defaultDtsInc / 90000.0 );
        }

        // If the TimeOut value has been specified.
        if( haveTimeOut )
        {
                // Set the TimeOut value in the object.
                enc->d_InputTimeOut = TimeOut;
        }
        else // The input time out was not specified.
        {
                // Use 2 seconds as the default.
                enc->d_InputTimeOut = 2.0;
        }

        // If the debug level has been specified.
        if( haveDbg )
        {
                enc->setDbgLev( dbgLev );
        }

        // The DtsOffset is the amount to add to the current output
        // time to arrive at an output time that avoids collisions
        // with data already held in the buffers waiting for output.
        // This needs to account for mux delay and is basically one
        // frame time plus an uncertainty time.
        enc->setEncDtsOffset( NORMAL_OUTPUT_DTS_STILL * enc->dtsInc() );

        // Set the port number in the object.
        enc->d_PortNumber = PortNumber;

        enc->pes()->setPesBitRate( bitRate );

        enc->setProgNum( progNum );

        // Set the stream type in the Encoder object.
        enc->setStreamType( streamType );

        return enc;
}

/*-------------------------------------------------------------
| NAME: TcpEnc::initialize
|-------------------------------------------------------------
|
| PURPOSE: To
|
| DESCRIPTION:
|
| EXAMPLE:
```

```
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
----------------------------------------------------------------*/
void
TcpEnc::initialize()
{
  // TBD -- scan the file to extract a PES record with a
  // PES sequence header so we can get the actual bitRate
  // and frame rate.
}

/*----------------------------------------------------------------
| NAME: TcpEnc::fillPesBuf
|----------------------------------------------------------------
|
| PURPOSE: To read one payload unit from the TCP socket into
|          the the Pes buffer.
|
| DESCRIPTION: See 'IctvMpeg.h'.
|
| A PES packet is constructed from the incoming data if it is
| not already in PES form.
|
| The PES packet is moved into the PES input buffer using
| 'addToPesBuf()'.
|
| When the PES packet is completed it is parsed into data
| fields held in the Pes object using the function
| 'scanPesHeader()'.
|
| Returns non-zero if a valid PES packet has been read in
| else return zero if there was an error.
|
| Attempts to resync packets if there are errors detected.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
|          [redacted] Fixed payload offset calculation to
|                     adjust for the size of the adaptation
|                     length field itself.
|          [redacted] Added support for Still Picture Format.
|          [redacted] More Still Picture Format revisions:
|                     duplicating picture header and
|                     incrementing temporal sequence field.
|          [redacted] Factored out P picture construction.
----------------------------------------------------------------*/
bool
TcpEnc::fillPesBuf()
{
        u8*     P;
        u8*     AtPayload;
        u8*     AtEnd;
        Int32   PayloadByteCount;
        UInt8   ID, A, B, C, D, E, F, G;
```

```
    UInt32  DataByteCount;
    Int32   i, j, TSPacketCount;
    u8              AdaptationFieldControl;
    BOOL    IsAdaptationField;
    u8*     AtPicture;
    u32     PPictureByteCount;

    static const UInt8 ptsHeader0[] =
    {
            0x00, 0x00, 0x01,                               // PES header start code
            Pes::MPEG_VIDEO_STREAM_ID_BASE, // stream ID for video
            0x00, 0x00,                                     // packet length
            0x84,                                           // marker=10,scram=00,
                                                    // pri=0, align=1,
                                                    // copyrt=0,orig=0
            0x80,                                           // PTS=1,DTS=0,ESCR=0,
                                                    // ESrate=0, trick=0,
                                                    // addl=0,CRC=0,ext=0
            0x05,                                           // PES header length
                                                    // (PTS only)
            0x21, 0x00, 0x01, 0x00, 0x01    // PTS (zero-value)
    };

    UInt8  ptsHeader[ sizeof(ptsHeader0) ];
    double StartTime;
    double TimeNow;
    BOOL   ok;

    // Clear the PES buffer.
    pes()->reset();

    // If debug info should be output.
    if( d_dbgLev & 2 )
    {
            // Get the current time before the frame is read in.
            StartTime = d_TimeStamp.getRelTime();
    }

    // Read the first six bytes of a 7 byte FIFO with
    // positions named 'A' thru 'G'.
    ok = ReceiveBytes( 6 );

    // If a read error happened.
    if( !ok )
    {
            // Just return with an error.
            return( 0 );
    }

    // Refer to the individual bytes read in.
    A = d_InputBuf[0];
    B = d_InputBuf[1];
    C = d_InputBuf[2];
    D = d_InputBuf[3];
    E = d_InputBuf[4];
F = d_InputBuf[5];

    // Until a valid packet prefix has been found.
    while( 1 )
    {
            // The next byte into the FIFO.
            ok = ReceiveBytes( 1 );
```

```
                // If a read error happened.
                if( !ok )
                {
                        // Just return with an error.
                        return( 0 );
                }

                G = *d_InputBuf;

                // Interpret the first byte in the FIFO as the packet ID.
                ID = A;

                // Interpret B, C, and D as the packet size value.
                DataByteCount =
                        ( ( (UInt32) B ) << 16 ) +
                        ( ( (UInt32) C ) <<  8 ) +
                        ( (UInt32) D );

                // If the packet has data and the size could be valid.
                if( DataByteCount && ( B < 2 ) )
                {
                        // If ID identifies the packet as PACKET_ID_ONE_PES_AS_TS
and
                        // E is a TS packet sync byte.
                        if( ID == PACKET_ID_ONE_PES_AS_TS &&
                        E  == SYNC_BYTE_VALUE )
                        {
                                // If the data length is an even multiple of the TS
                                // packet size.
                                if( ( DataByteCount % BYTES_PER_TS_PACKET ) == 0 )
                                {
                                        // Assume a valid TS type packet follows.

                                        // Get the remainder of the packet and
process it.
                                        goto GetRestOfPacket;
                                }
                        }

                        // Not a TS type packet but the other kinds haven't
                        // be eliminated yet.

                        // If E is the first byte of a PES packet start code prefix.
                        if( E == 0 && F == 0 && G == 1 )
                        {
                                // Assume a valid packet of some type follows.

                                // Get the remainder of the packet and process it.
                                goto GetRestOfPacket;
                        }
                }

                // Shift the bytes in the input FIFO.
                A = B; B = C; C = D; D = E; E = F; F = G;

        } // End of while loop.

//////////////////
GetRestOfPacket:// Get the remainder of the packet the input buffer.
//////////////////

        // Read in the data remembering that we already have the first
        // three bytes in E, F and G.
```

```
ok = ReceiveBytes( DataByteCount - 3 );

// If a read error happened.
if( !ok )
{
        // Just return with an error.
        return( 0 );
}

// At this point a packet is in the input buffer and it
// has been partially qualified.

// If the ID code is not recognized.
if( ID != PACKET_ID_ONE_PES        &&
    ID != PACKET_ID_ES_OF_ONE_PES &&
    ID != PACKET_ID_ONE_PES_AS_TS &&
    ID != PACKET_ID_STILL_PICTURE &&
    ID != PACKET_ID_ACTIVATE      &&
    ID != PACKET_ID_DEACTIVATE )
{
        // Could put more validation here.

        // Interpret the packet as an elementary stream.
        ID = PACKET_ID_ES_OF_ONE_PES;
}

// Depending on the format of the input packet.
switch( ID )
{
        case PACKET_ID_ONE_PES: // PES packet
        {
                // Move the first three bytes to the Pes buffer:
                // these are the Start Code bytes.
                addToPesBuf( &E, 1 );
                addToPesBuf( &F, 1 );
                addToPesBuf( &G, 1 );

                // Move the rest of the PES packet to the Pes buffer.
                addToPesBuf( d_InputBuf, DataByteCount - 3 );

                break;
        }

        // Still Picture prefix format.
        case PACKET_ID_STILL_PICTURE:
        {
                // Move the first three bytes to the Pes buffer:
                // these are the Start Code bytes.
                addToPesBuf( &E, 1 );
                addToPesBuf( &F, 1 );
                addToPesBuf( &G, 1 );

                // Clear the PES header length fields: remember to
                // subtract 3 for the PES header bytes in E, F and G.
                d_InputBuf[ PES_PACKET_LENGTH_HIBYTE - 3 ] = 0;
                d_InputBuf[ PES_PACKET_LENGTH_LOBYTE - 3 ] = 0;

                // Refer to end of the data in the input buffer.
                AtEnd = d_InputBuf + DataByteCount - 3;

                // If dumping PES packet data to a log file.
                if( d_dbgLev == 10 )
                {
```

```
                // Write the ICTV prefix header.
                fwrite( &A, (u32) 1, (u32) 1, d_LogFile );
                fwrite( &B, (u32) 1, (u32) 1, d_LogFile );
                fwrite( &C, (u32) 1, (u32) 1, d_LogFile );
                fwrite( &D, (u32) 1, (u32) 1, d_LogFile );

                // Write the first three bytes.
                fwrite( &E, (u32) 1, (u32) 1, d_LogFile );
                fwrite( &F, (u32) 1, (u32) 1, d_LogFile );
                fwrite( &G, (u32) 1, (u32) 1, d_LogFile );

                // Write out the rest of the bytes.
                fwrite( d_InputBuf,
                        DataByteCount - 3,
                        (u32) 1,
                        d_LogFile );
        }

        // Find the picture header in the input buffer.
        AtPicture =
                ToSpecificStartCode(
                        d_InputBuf,
                            // Address of the byte where the search
                // begins.
                //
                        AtEnd,
                            // Address of the first byte after
                        // the last valid byte in the range.
                //
                        (u32) PICTURE_START_CODE );
                                // The value of the fourth byte of
                                // the target Start Code.

        // If no picture was found in the source data.
        if( ! AtPicture )
        {
                // Then return with an error.
                return( 0 );
        }

        for( j = 0; j < 1; j++ )
        {
                // Append a sequence end code.
                *AtEnd++ = 0;
                *AtEnd++ = 0;
                *AtEnd++ = 1;
                *AtEnd++ = SEQUENCE_END_CODE;
        }

#if 1 // 12.01.99 TL Removed this section of code which is
      // called for in the GI spec but which doesn't work:
      // it could be that some parts of the dummy P frame
      // are not being constructed properly -- more work
      // needed here to understand why this doesn't work
      // as specified.

        // Make a P picture that repeats the I picture.
        PPictureByteCount =
                MakeDummyCodedPFrame(
                        AtPicture, // The first byte of the Picture
                                   // Header of the I Picture used
                                   // as the source.
```

```
                                //
                                AtEnd,      // The first byte after the last
                                            // byte in the I Picture used as
                                            // the source.
                            //
                                AtEnd );    // The destination buffer where
                                            // the P Picture will be
constructed.

                        // Advance the end-of-data address for the P frame
                        // data.
                        AtEnd += PPictureByteCount;

                        for( j = 0; j < 1; j++ )
                        {
                                // Append a sequence end code.
                                *AtEnd++ = 0;
                                *AtEnd++ = 0;
                                *AtEnd++ = 1;
                                *AtEnd++ = SEQUENCE_END_CODE;
                        }
#endif // 0
                        // Move the rest of the PES packet to the Pes buffer.
                        addToPesBuf( d_InputBuf, AtEnd - d_InputBuf );

                        break;
                }

                case PACKET_ID_ES_OF_ONE_PES: // ES packet
                {
                        // We need to first build a PES header with a PTS
                        // field since this is missing.

                        // Modify a standard copy of the PES header.
                        memcpy( ptsHeader, ptsHeader0, sizeof(ptsHeader) );

                        // If the PES packet length can fit into 16 bits.
                        if( ( DataByteCount & 0xffff ) == DataByteCount )
                        {
                                // Set the high byte of the PES packet length
                                // field.
                                ptsHeader[ Pes::PES_PACKET_LENGTH_INDEX ] =
                                        (UInt8) ( DataByteCount >> 8 );

                                // Set the low byte of the PES packet length
                                // field.
                                ptsHeader[ Pes::PES_PACKET_LENGTH_INDEX + 1 ] =
                                        (UInt8) DataByteCount;
                        }

                        // Set the steamid in the PES packet header based
                        // on the configured stream type.
                        // TBD - WRONG: streamType and StreamID are two
                        // different things.
                        //ptsHeader[ STREAM_ID_INDEX ] = streamType();

                        // The correct value for the PTS will be inserted
                        // when 'TcpEnc::threadStep()' is called.

                        // Insert the PES header into the PES buffer.
                        addToPesBuf( ptsHeader, sizeof(ptsHeader) );

                        // Move the first ES three bytes to the Pes buffer.
```

```
                addToPesBuf( &E, 1 );
                addToPesBuf( &F, 1 );
                addToPesBuf( &G, 1 );

                // Move the rest of the ES segment to the Pes buffer.
                addToPesBuf( d_InputBuf, DataByteCount - 3 );

                break;
        }

        case PACKET_ID_ONE_PES_AS_TS: // TS packets
        {
                // Calculate the number of TS packets in the input
                // buffer.
                TSPacketCount = DataByteCount / BYTES_PER_TS_PACKET;

                // The first three bytes of the first TS packet are
                // in E, F and G.  Refer to where E would be if it
                // were part of the first TS packet.
                P = d_InputBuf - 3;

                // For each TS packet in the input buffer.
                for( i = 0; i < TSPacketCount; i++ )
                {
                        // Get the value of the adaptation field control
field.
                        AdaptationFieldControl = ( P[ ADAPT_CTRL_INDEX ] >>
4 ) & 3;

                        // Determine if there is an adaptation field.
                        IsAdaptationField = AdaptationFieldControl & 2;

                        // Calculate the beginning of the first payload byte

                        // of the current TS packet.

                        // Refer to the where the first payload byte would
be
                        // in the TS packet if there were no adaptation
field.
                        AtPayload = P + ADAPT_LEN_INDEX;

                        // If there is an adaptation field.
                        if( IsAdaptationField )
                        {
                                // Adjust the payload field address to
account
                                // for the size of the adaptation field,
adding one
                                // for the size of the length field itself.
                                AtPayload += P[ ADAPT_LEN_INDEX ] + 1;
                        }

                        // The size of the payload section of the TS packet
                        // is calculated by subtracting the offset of the
                        // first payload byte from the overall TS packet
size.
                        PayloadByteCount = BYTES_PER_TS_PACKET - ( AtPayload
- P );

                        // Move the payload bytes to the Pes buffer.
                        addToPesBuf( AtPayload, PayloadByteCount );
```

```
                        // Advance the TS packet pointer.
                        P += BYTES_PER_TS_PACKET;
                }

                break;
        }
}

// Get the current time after reading a complete frame.
d_CurrentFrameArrivalTime = d_TimeStamp.getRelTime();

// If debug info should be output.
if( d_dbgLev & 2 )
{
        // Report the time it took to read the frame.
        cout << Util::formatFixed(d_CurrentFrameArrivalTime, 5) <<
        " TcpEnc::fillPesBuf: Time to read frame: " <<
        Util::formatFixed(d_CurrentFrameArrivalTime - StartTime, 5) <<
        " secs" << endl;
}

// Parse the data in the PES header into fields in
// the Pes object.
pes()->scanPesHeader();

// If debug info should be output.
if( d_dbgLev & 2 )
{
        cout << "TcpEnc::fillPesBuf: Packet ID=" << ( (UInt32) ID )<<
        "DataByteCount= " << DataByteCount << " bytes" <<
        endl;

        cout << "TcpEnc Pes:  pts=" <<
        Util::formatFixed(pes()->pts()) <<
        " dts=" << Util::formatFixed(pes()->dts()) <<
        " dtsInc=" << dtsInc() << endl;
}

// Every 24 frames.
if( (d_dbgLev & 1) && (d_InputFrameCount % 24) == 0 )
{
        // Get the current time after reading a complete frame.
        TimeNow = d_TimeStamp.getRelTime();

        // Report the time it took to read the frame.
        cout << Util::formatFixed(TimeNow, 5) <<
        " " << d_BytesReceived <<
        " frame# " << d_InputFrameCount << endl;

        cerr << Util::formatFixed(TimeNow, 5) <<
        " " << d_BytesReceived <<
        " frame# " << d_InputFrameCount << endl;
}

// Count how many PES data bytes have been received total.
d_totalPesData += pes()->inPosn();

// Enter or exit Still Frame Mode depending on the packet
// just received.
d_IsStillFrameMode = ( ID == PACKET_ID_STILL_PICTURE );

// The DtsOffset is the amount to add to the current output
// time to arrive at an output time that avoids collisions
```

```
        // with data already held in the buffers waiting for output.
        // This needs to account for mux delay and is basically one
        // frame time plus an uncertainty time.

        // Adjust the normal dtsoffset depending on whether in
        // still frame or regular mode.
        if( d_IsStillFrameMode )
        {
                // More time is needed on average for streams with
                // only I Pictures.
                setEncDtsOffset( NORMAL_OUTPUT_DTS_STILL * dtsInc() );
        }
        else
        {
                // Two frame times is adequate for streams with
                // I and P Pictures.
                setEncDtsOffset( NORMAL_OUTPUT_DTS * dtsInc() );
        }

        // Count how many PES packets have been received total.
        d_InputFrameCount++;

        return( true );
}

/*-------------------------------------------------------------
| NAME: TcpEnc::GetNextFrame
|-------------------------------------------------------------
|
| PURPOSE: To read the next frame from the input or re-use
|          the last frame if no input data is available.
|
| DESCRIPTION: Also monitors how long it's been since the
| last input so that the connection can be closed on a
| timeout.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
|          [redacted] Fixed time out.
-------------------------------------------------------------*/
void
TcpEnc::GetNextFrame()
{
        double TimeNow;
        double ElapsedTimeInSeconds;
        double TimeOutSeconds;

        // If connected and no data available.
        if( d_IsConnected && d_BytesAvailable == 0 )
        {
                // Get the current time.
                TimeNow = d_TimeStamp.getRelTime();

                // Calculate the elapsed time in seconds.
                ElapsedTimeInSeconds = TimeNow - d_LastInputTime;

                // If this is the first frame.
                if( d_IsFirstFrame )
```

```
        {
                // Allow five times the normal time out.
                TimeOutSeconds = d_InputTimeOut * 5.0;
        }
        else // This is not the first frame.
        {
                // Use the normal time out period.
                TimeOutSeconds = d_InputTimeOut;
        }

        // If the expected data takes longer than the
        // read timeout period.
        if( ElapsedTimeInSeconds > TimeOutSeconds )
        {
                // Close the TCP connection.
                CloseConnection();
        }
}

// If this is still frame mode.
if( d_IsStillFrameMode )
{
        // If there is a current frame and there are bytes
        // available for reading from the socket.
        if( d_IsCurrentFrameAvailable &&
            d_IsConnected             &&
            d_BytesAvailable )
        {
                // Then clear the frame available flag so that
                // a new frame will be read rather than repeating
                // the current frame.
                d_IsCurrentFrameAvailable = 0;
        }

        // If we can re-use the current frame.
        if( d_IsCurrentFrameAvailable )
        {
                // Return treating the previous frame as repeat frame.
                return;
        }
}

// If there is no current frame to process.
if( ! d_IsCurrentFrameAvailable )
{
        // If data is available to be read.
        if( d_IsConnected && d_BytesAvailable )
        {
                // Try to receive one frame.
                d_IsCurrentFrameAvailable = fillPesBuf();
    }
}

// If a valid frame still has not been received.
if( ! d_IsCurrentFrameAvailable )
{
        // Return without doing anything else.
        return;
}

// If the current frame should be treated as if it
// were the very first frame ever processed.
if( d_IsFirstFrame )
```

```
        {
                // If this not Still Frame Mode.
        if( ! d_IsStillFrameMode )
        {
                // If the current frame doesn't have a PTS value.
                if( ! pes()->ptsFlag() )
                {
                        // Then discard the current frame.
                        d_IsCurrentFrameAvailable = 0;
                }
        }
    }
}

/*-------------------------------------------------------------
 NAME: TcpEnc::ReceiveBytes
-------------------------------------------------------------

 PURPOSE: To receive a specified number of bytes to the
          input buffer.

 DESCRIPTION: Returns only after the specified number of
 bytes has been received from the TCP socket.

 The input data is placed at the beginning of the input
 buffer, 'd_InputBuf'.

 If the input buffer is too small to accomodate the required
 data it is reallocated.

 EXAMPLE:

 NOTE:

 ASSUMES: There is no data currently in the input buffer.

 HISTORY: [redacted]
          [redacted] Added error return flag and reporting.
          [redacted] Added use of Socket_Data_Avail so that
                     disconnections can be detected without
                     hanging in a blocking read.
-------------------------------------------------------------*/
BOOL
TcpEnc::ReceiveBytes( u32 BytesToRead ) // How many bytes to read.
{
        u8*             AtBuf;
        u32     BytesReadThisPass;
        double  TimeNow;
        double  ElapsedTime;
        s32     BytesAvailable;

        // If the SOCKET isn't connnected.
        if( ! d_IsConnected )
        {
                // Return failure
                return( 0 );
        }

        // If the expected number of bytes is more than the
        // space available in the input buffer.
        if( BytesToRead > d_InputBufSize )
        {
                // If there is an input buffer to free.
```

```
        if( d_InputBuf )
        {
                // Deallocate the input buffer.
                free( d_InputBuf );
        }

        // Mark the input buffer as missing.
        d_InputBuf = 0;
}

// If the input buffer is missing.
if( ! d_InputBuf )
{
        // Allocate an input buffer large enough for the
        // expected bytes.
        d_InputBuf = (u8*) malloc( BytesToRead );

        // Update the size of the input buffer.
        d_InputBufSize = BytesToRead;
}

// Refer to the input buffer's first byte.
AtBuf = d_InputBuf;

// Until the required number of bytes have been received.
while( BytesToRead )
{
        // Get how many bytes are currently available.
        BytesAvailable = (s32) Socket_Data_Avail( d_Socket );

        // If there was an error testing for available data.
        if( BytesAvailable < 0 )
        {
                // Close down the connection.
                goto CloseSocketOnError;
        }
        else // No error testing for available data.
        {
                // If there is data available to read.
                if( BytesAvailable )
                {
                        // If there are more bytes available than
                        // are needed.
                        if( BytesAvailable > BytesToRead )
                        {
                                // Only read what is required.
                                BytesAvailable = BytesToRead;
                        }

                        // Read as many remaining bytes as are
                        // available to the input buffer.
                        BytesReadThisPass =
                                Socket_Read( d_Socket, AtBuf, BytesAvailable
);

                        // If the socket quits gracefully or there was a
                        // read error.
                        if( ( BytesReadThisPass == 0 ) ||
                            ( BytesReadThisPass == SOCKET_ERROR ) )
                        {
                                // Close down the connection.
                                goto CloseSocketOnError;
                        }
```

```
                        else // Some data was read.
                        {
                                // Note the time when the data arrived.
                                d_LastInputTime = d_TimeStamp.getRelTime();

                                // Account for the bytes read.
                                BytesToRead -= BytesReadThisPass;

                                // Advance the destination in the buffer.
                                AtBuf += BytesReadThisPass;

                                // Update the running total for the encoder.
                                d_BytesReceived += BytesReadThisPass;
                        }
                }
                else // No data currently available.
                {
                        // Sleep ten milliseconds.
                        Sleep( 10 );

                        // Get the current time.
                        TimeNow = d_TimeStamp.getRelTime();

                        // Calculate the elapsed time in seconds.
                        ElapsedTime = TimeNow - d_LastInputTime;

                        // If the expected data takes longer than the
                        // read timeout period.
                        if( ElapsedTime > d_InputTimeOut )
                        {
                                // Close down the connection.
                                goto CloseSocketOnError;
                        }
                }
        }
    }

    // All bytes requested have been read.
    return( 1 );

//////////////////////
CloseSocketOnError://
//////////////////////

    // If debug info should be output.
    if( d_dbgLev & 2 )
    {
        // Report the error.

        // Report the time it took to read the frame.
        cout <<
        " TcpEnc::ReceiveBytes error: Closing connection."
         << endl;

        cerr <<
        " TcpEnc::ReceiveBytes error: Closing connection."
         << endl;
    }

    // Close the network socket.
    CloseConnection();

    // Return 0 to indicate a read error.
```

```
        return( 0 );
}

/*-------------------------------------------------------------
| NAME: TcpEnc::ResetOutputDts
|-------------------------------------------------------------
|
| PURPOSE: To reset the value used to schedule when frames
|          can enter the the mux input queue.
|
| DESCRIPTION: The OutputDts is the earliest time when a frame
| can leave an encoder's waiting  queue for input into the
| mux.
|
| STC is the point in time where a byte is leaving the mux.
|
| Initialize the OutputDTS to be the current STC adjusted by
| the amount of time it takes a frame to flow through the
| mux, phase adjusted to an integral frame time.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES: STC time refers to the byte currently leaving
|          the mux.
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
void
TcpEnc::ResetOutputDts()
{
        double Stc, DtsOffset, OutputDts, DtsInc;

        // Get local copies of the values we need.
        Stc       = Util::stcTime();
        DtsOffset = dtsOffset();
        DtsInc    = dtsInc();

        // Calculate the normal outputDts.
        OutputDts = Stc + DtsOffset;

        // FRAME PHASE NORMALIZATION RULE: The value of OutputDts
        // is used as a basis for the PTS and DTS timestamps
        // which must be in terms of integral frame times.  This
        // next section is responsible for snapping the DtsOffset
        // to the nearest integral frame

        // Calculate the nearest integral frame time.
        OutputDts =
                floor( ( OutputDts + (DtsInc * .5) ) / DtsInc ) * DtsInc;

        // Set the new outputDts.
        setOutputDts( OutputDts );
}

/*-------------------------------------------------------------
| NAME: TcpEnc::setQuitRequest
|-------------------------------------------------------------
|
| PURPOSE: To
|
| DESCRIPTION:
```

```
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
void
TcpEnc::setQuitRequest( bool rhs )
{
        DmThread::setQuitRequest(rhs);
}

/*-------------------------------------------------------------
| NAME: TcpEnc::start
|-------------------------------------------------------------
|
| PURPOSE: To
|
| DESCRIPTION:
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
void
TcpEnc::start()
{
        threadCreate();
}

/*-------------------------------------------------------------
| NAME: TcpEnc::stop
|-------------------------------------------------------------
|
| PURPOSE: To
|
| DESCRIPTION:
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
bool
TcpEnc::stop()
{
        bool rslt = true;

        if( running() )
        {
                rslt = stopThread();
        }
```

```
        // TBD
        return rslt;
}

/*-------------------------------------------------------------
| NAME: TcpEnc::threadInit
|-------------------------------------------------------------
|
| PURPOSE: To
|
| DESCRIPTION:
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
void
TcpEnc::threadInit()
{
        ;
}

/*-------------------------------------------------------------
| NAME: TcpEnc::threadStep
|-------------------------------------------------------------
|
| PURPOSE: To convert one PES packet to TS packets, feed
|          them into the multiplexer and read another PES
|          packet.
|
| DESCRIPTION: Inserts current PTS/DTS if the PES packet has
| those fields.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] From Ac3Enc and IbmDemoEnc.cpp.
|          [redacted] Added special Still Frame Mode handling
|                     for the PTS.
|          [redacted] Factored out socket connection code.
|          [redacted] Added frame repeat for still frame mode.
-------------------------------------------------------------*/
void
TcpEnc::threadStep()
{
        // Wait for and make TCP connection if possible.
        WaitForConnection();

        // Measure how many bytes are currently available from
        // the TCP socket.
        d_BytesAvailable = (s32) Socket_Data_Avail( d_Socket );

        // If there was an error testing for available data.
        if( d_BytesAvailable < 0 )
        {
                // Then there is no data available.
```

```
                d_BytesAvailable = 0;

                // Close the TCP connection.
                CloseConnection();

                // Don't return here because we may want to keep
                // repeating the last frame we have.
        }

        // Read the next frame to send.
        GetNextFrame();

        // If there is a current frame to process.
        if( d_IsCurrentFrameAvailable )
        {
                // Wait for next frame time.
                WaitForNextFrameTime();

                // Keep the time scheduled for inputing the current
                // frame into the mux within bounds.
                NormalizeOutputDts();

                // At this point we have a valid frame and the outputDts value
                // applies to the current frame.

                // Adjust the any PTS/DTS values in the current PES frame.
                AdjustFrameTimeStamps();

                // Convert PES buffer to TS packets and feed them into
                // the wait queue that feeds the mux input.
                enqueuePesBuf();

                // Count how many frames have been queued for output.
                d_OutputFrameCount++;

                // If this is not still frame mode.
                if( ! d_IsStillFrameMode )
                {
                        // Clear the frame availability flag so that the
                        // same frame won't be sent multiple time.
                        d_IsCurrentFrameAvailable = 0;
                }

                // Clear the first frame flag since we've completed
                // all initialization required if this was the first
                // frame.
                d_IsFirstFrame = 0;
        }
}

/*-------------------------------------------------------------
| NAME: TcpEnc::showSetup
|-------------------------------------------------------------
|
| PURPOSE: To display setup and statistics as text.
|
| DESCRIPTION:
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
```

```
|
| HISTORY: [redacted]
-------------------------------------------------------------*/
void
TcpEnc::showSetup( ostream &strm ) const
{
        // If the output stream is valid.
        if( strm.opfx() )
        {
                strm << "{class TcpEnc " << threadName() << ":";
        strm << "  Reading " << ( d_isVideo ? " Video" : " Audio" )
                        << " PES packets from port=" << d_PortNumber <<
                        " output PID=" << esPid();
                strm << endl;

                // Show the generic encoder setup.
                Encoder::showSetup(strm);

                strm << "}" << endl;

                // Flush the output stream.
                strm.osfx();
        }
}

/*-------------------------------------------------------------
| NAME: TcpEnc::showStats
|-------------------------------------------------------------
|
| PURPOSE: To
|
| DESCRIPTION:
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Added line for number of packets sent.
-------------------------------------------------------------*/
void
TcpEnc::showStats(ostream &strm) const
{
        // If the stream is valid.
        if( strm.opfx() )
        {
                strm << "TcpEnc " << threadName() << ":";
        strm << "  Read " << d_InputFrameCount << " PES packets( "
                 << Util::formatFixed(d_totalPesData/double(1<<20),3)
                 << " MB)" << endl;

                strm << "TcpEnc " << threadName() << ":";
        strm << "  Sent " << d_OutputFrameCount <<
        " PES packets" << endl;

                Encoder::showStats(strm);

                // Flush the output stream.
                strm.osfx();
        }
}
```

```
/*--------------------------------------------------------------
| NAME: TcpEnc::WaitForConnection
|--------------------------------------------------------------
|
| PURPOSE: To wait for an incoming TCP connection, make
|          the connection and prepare for frame data.
|
| DESCRIPTION: If the socket is already connected then nothing
| is done.
|
| If not connected then:
|
| 1. Enter the state where no frames have yet been processed.
|
| 2. Initialize and bind a socket.
|
| 3. Configure the socket as blocking.
|
| 4. Configure a 128K receive buffer instead of the default
|    8K buffer.
|
| 5. Wait for and accept an incoming connection on the
|    configured port for the TcpEnc instance.
|
| 6. On connection enter the connected state.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] Factored out of TcpEnc::threadStep().
|          [redacted] Added default to Still Frame Mode on
|                     connection.
|          [redacted] Added opening of special log file if
|                     debug level is 9.
--------------------------------------------------------------*/
void
TcpEnc::WaitForConnection()
{
        int                             status;
        unsigned long   blocking;
        s32                     Err;

        // If debug level is 9 and LOG_RECORD_COUNT frames have
        // been collected.
        if( d_dbgLev == 9 && d_LogRecordCount == LOG_RECORD_COUNT )
        {
                // Disconnect and output the data.
                CloseConnection();
        }

        // If the TCP socket is not connected.
        if( d_IsConnected == 0 )
        {
                // Initialize a TCP socket and bind it to the
                // configured port number.
                d_ListenSocket =
                        Socket_Init_And_Bind( (Int32) SOCK_STREAM, d_PortNumber );

                // If there was an error creating or binding the
```

```
                // socket.
                if( d_ListenSocket <= 0 )
                {
                        // If debug info should be output.
                        if( d_dbgLev )
                        {
                                cout << "WaitForConnection error: Cannot bind port
number " <<
                                d_PortNumber << endl;

                                cerr << "WaitForConnection error: Cannot bind port
number " <<
                                d_PortNumber << endl;
                        }

                        // Return without connecting.
                        return;
                }

                // Make the listening socket a blocking socket.

                // Set 'blocking' to zero to indicate that the socket should
                // be blocking.
                blocking = 0;
                status = ioctlsocket( d_ListenSocket, (long) FIONBIO, &blocking );

                // If 'int' is less than four bytes long.
                if( sizeof( int ) < 4 )
                {
                        // Make a 30000 byte kernel TCP receive buffer for the
socket.
                        Err = Socket_SetSockOpt( d_ListenSocket, SOL_SOCKET,
SO_RCVBUF, 30000 );
                }
                else
                {
                        // Make a 128K byte kernel TCP receive buffer for the
socket.
                        Err = Socket_SetSockOpt( d_ListenSocket, SOL_SOCKET,
SO_RCVBUF, 128 * 1024 );
                }

                // If there was an error changing the buffer size.
                if( Err )
                {
                        // Close the code to release OS resources.
                        Socket_Close( d_ListenSocket );

                        // Mark our socket as missing.
                        d_ListenSocket = 0;

                        // If debug info should be output.
                        if( d_dbgLev )
                        {
                                // Report the error.
                                cout << "WaitForConnection error setting rcv buffer
size " << endl;

                                cerr << "WaitForConnection error setting rcv buffer
size " << endl;
                        }

                        // Return without connecting.
                        return;
```

```
	}

	// Wait here for an incoming connection.
	d_Socket = Socket_Listen_Accept( d_ListenSocket, 0 );

	// If a valid connection was made.
	if( d_Socket )
	{
		// Do the buffers for this socket also need to be
reconfigured?

		// Note the socket connection status.
		d_IsConnected = 1;

		// Treat the connection as an input for time out purposes.
		d_LastInputTime = d_TimeStamp.getRelTime();

		// Then treat the next frame as if it were the first
		// frame ever processed.
		d_IsFirstFrame = 1;

		// Then there is no valid current frame available.
		d_IsCurrentFrameAvailable = 0;

		// Default to not being in still frame mode.
		d_IsStillFrameMode = 0; // 1;

		// If the debug level is 9.
		if( d_dbgLev == 9 )
		{
			// Create a buffer big enough to hold data for
			// log records.
			d_LogBuffer = (u8*)
				malloc( LOG_RECORD_COUNT *
					sizeof( TcpLogRecord ) );

			// Set the end of buffer marker.
			d_LogBufferEnd = d_LogBuffer;

			// Initially there are no records.
			d_LogRecordCount = 0;
		}
		else // No log buffer.
		{
			// Clear the log file field.
			d_LogBuffer    = 0;
			d_LogBufferEnd = 0;
		}

		// If dumping PES packet data to a log file.
		if( d_dbgLev == 10 )
		{
			// Then open a special log file.
			d_LogFile = fopen( "TcpLog.pes", "wb" );
		}
	}
	else // A connection error occurred.
	{
		// Close the listening socket.
		Socket_Close( d_ListenSocket );

		// Mark our socket as missing.
		d_ListenSocket = 0;
```

```
                        // If debug info should be output.
                        if( d_dbgLev )
                        {
                                cout << "threadInit error: Cannot accept on port
number " <<
                                d_PortNumber << endl;

                                cerr << "threadInit error: Cannot accept on port
number " <<
                                d_PortNumber << endl;
                        }

                        // Return without connecting.
                        return;
                }
        }
}

/*-------------------------------------------------------------
| NAME: TcpEnc::WaitForNextFrameTime
|-------------------------------------------------------------
|
| PURPOSE: To meter mux input and sync the outputDTS counter
|          to the STC.
|
| DESCRIPTION: This procedure regulates the input of a frame
| into the mux and also as an indirect result it syncs the
| outputDTS counter to the STC counter through the amount of
| time used to make the thread sleep.
|
| Uses expontial moving averages of the STC and OutputDts
| to smooth out short term sample fluctuations.
|
| EXAMPLE:
|
| NOTE:
|
| ASSUMES: STC time refers to the byte currently leaving
|          the mux.
|
| HISTORY: [redacted] Copied from SimEnc::threadStep.
|                     Cleaned up initial frame condition.
-------------------------------------------------------------*/
void
TcpEnc::WaitForNextFrameTime()
{
        f64         Stc, DtsOffset;
        f64         Odt, Wait, ActualWait;
        static f64  StcX, OdtX;
        f64         Wt;

        // Set the smoothing weight depending on the output
        // frame number: initially very responsive to short
        // term differences and later less so.
        if( d_OutputFrameCount < 31 )
        {
                Wt = .5;
        }
        else // At least 30 frames have been output.
        {
                Wt = .1;
        }
```

```
    // Get values we need to local variables.
    Stc       = Util::stcTime();
    Odt       = outputDts();
    DtsOffset = dtsOffset();

    // If the current frame is the first frame.
    if( d_IsFirstFrame )
    {
            // Set the initial outputDts value.
            ResetOutputDts();

            // Get the current outputDts value.
            Odt = outputDts();

            // Set the initial smoothed values to the
            // actual values.
            StcX = Stc;
            OdtX = Odt;
    }
    else // Not the first frame.
    {
            // Update the smoothed values.
            StcX = ( Wt * Stc ) + ( ( 1.0 - Wt ) * StcX );
            OdtX = ( Wt * Odt ) + ( ( 1.0 - Wt ) * OdtX );
    }

    // We need to calculate how long it will be before
    // the next frame can be queued: OutputDts was
    // incremented the last time enqueuepesbuf() was
    // called so it refers to the time when the next
    // frame should enter the mux from the waiting
    // buffer.
    //Wait = Util::timeDiff( Odt - ( Stc + DtsOffset ) );
    Wait = Util::timeDiff( OdtX - ( StcX + DtsOffset ) );

    // If the thread should delay.
if( Wait > 0.0 )
    {
            // Wait until it's time to input the current frame.
            threadSleep( Wait );
    }

    // If log records are being collected.
    if( d_dbgLev == 9  )
    {
            // Calculate the actual wait time.
            ActualWait = Util::timeDiff( Util::stcTime() - Stc );

            // Collect the data for the current frame.
            TheLogRecord.Stc              = Stc;
            TheLogRecord.RequestedWait    = Wait;
            TheLogRecord.ActualWait       = ActualWait;
            TheLogRecord.OutputDts        = Odt;
            TheLogRecord.InputFrameNumber = d_InputFrameCount;

            // Append the log record to the log buffer.
            memcpy( d_LogBufferEnd,
                    (u8*) &TheLogRecord,
                    sizeof( TcpLogRecord ) );

            // Account for the new record just added.
            d_LogBufferEnd   += sizeof( TcpLogRecord );
```

```
                d_LogRecordCount += 1;
        }
}

/*-------------------------------------------------------------
| NAME: ToSlice
|-------------------------------------------------------------
|
| PURPOSE: To find the address of the next picture Slice in a
|          range of bytes.
|
| DESCRIPTION: Returns the address of the first byte of the
| Start Code of the next Slice header in a buffer or zero
| if no slice code was found.
|
| EXAMPLE:
|
|        AtStartCode = ToSlice( Lo, Hi );
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] from 'ToNextStartCode()'.
-------------------------------------------------------------*/
                                        // Returns the address of the
                                        // first byte of the slice start
u8*                                 // code or zero if none was found.
ToSlice(
        u8* Lo,         // Address of the byte where the search
                        // begins.
                        //
        u8* Hi )        // Address of the first byte after
                        // the last valid byte in the range.
{
        u8 A, B, C, D;

        // Get the first three bytes.
        A = *Lo++;
        B = *Lo++;
        C = *Lo++;

        // As long as the current byte is valid.
        while( Lo < Hi )
        {
                // Get the value of the fourth byte.
                D = *Lo++;

                // If A, B, and C match the first bytes of
                // a start code and the fourth byte matches
                // a slice code value.
                if( ( A == 0 ) &&
                    ( B == 0 ) &&
                    ( C == 1 ) &&
                    ( D >= (u8) FIRST_SLICE_START_CODE ) &&
                    ( D <= (u8) LAST_SLICE_START_CODE ) )
                {
                        // Then we have found the start code.

                        // Return the address of the first
                        // byte of the start code.
                        return( Lo - 4 );
                }
```

```
                // Shift the bytes in the input FIFO.
                A = B; B = C; C = D;
        }

        // No start code was found.

        // Return zero.
        return( 0 );
}

/*-------------------------------------------------------------
| NAME: ToSpecificStartCode
|-------------------------------------------------------------
|
| PURPOSE: To find the address of the next matching MPEG Start
|          Code in a buffer.
|
| DESCRIPTION: Returns the address of the first byte of the
| next matching Start Code or zero if none was found.
|
| EXAMPLE:
|
|        AtStartCode =
|            ToSpecificStartCode( AtBuf,
|                                  AfterBuf,
|                                  MatchValue );
|
| NOTE:
|
| ASSUMES:
|
| HISTORY: [redacted] from 'ToNextStartCode()'.
-------------------------------------------------------------*/
                                        // Returns the address of the
                                        // first byte of the target start
u8*                             // code or zero if none was found.
ToSpecificStartCode(
        u8* Lo,         // Address of the byte where the search
                        // begins.
                        //
        u8* Hi,         // Address of the first byte after
                        // the last valid byte in the range.
                        //
        u32 MatchValue )// The value of the fourth byte of
                        // the target Start Code.

{
        u8 A, B, C, D;

        // Get the first three bytes.
        A = *Lo++;
        B = *Lo++;
        C = *Lo++;

        // As long as the current byte is valid.
        while( Lo < Hi )
        {
                // Get the value of the fourth byte.
                D = *Lo++;

                // If A, B, and C match the first bytes of
                // a start code and the fourth byte matches
```

```
                // the target value.
                if( ( A == 0 ) &&
                    ( B == 0 ) &&
                    ( C == 1 ) &&
                    ( D == (u8) MatchValue ) )
                {
                        // Then we have found the start code.

                        // Return the address of the first
                        // byte of the start code.
                        return( Lo - 4 );
                }

                // Shift the bytes in the input FIFO.
                A = B; B = C; C = D;
        }

        // No start code was found.

        // Return zero.
        return( 0 );
}

//---------------------------------------
```

EXHIBIT F

Arctos Version 1.0

Functional Specification

ICTV, Inc.
14600 Winchester Blvd.
Los Gatos, CA 95032
(408) 364-9200

Approval 1	_____	Title:	_____	Date:	_____
Approval 2	_____	Title:	_____	Date:	_____
Approval 3	_____	Title:	_____	Date:	_____
Approval 4	_____	Title:	_____	Date:	_____

Revised On:	[redacted]
Document Number:	FS0002-01.01
Author:	Arctos Team

1. Introduction

This document describes the features for the ARCTOS Information Service.

1.1 Purpose and Scope

This document describes the functional requirements, design, testing, and planning issues necessary to properly release the Arctos project. Both engineering and QA will use this document as a guide for implementation and verification of Arctos project. All details contained in this specification should be followed exactly. If it is discovered during implementation that changes to these details are required, the changes must be reviewed, and this document modified to reflect the actual differences. In addition, no functionality other than that contained in this document will be implemented without the same review process.

1.2 Related Documents

As other documents are referenced within the body of this document, they will be included below for reference.

Document #	Publisher	Title	Author
TS001-01	ICTV	Arctos Test Specification	Arctos Team.

1.3 Omissions and Uncertainties

None.

1.4 Confidentiality

This document is the confidential proprietary property of ICTV, Inc. The information contain herein may not be disclosed, copied, distributed or otherwise utilized without the express written permission of ICTV, Inc.

1.5 Methodology and Conventions

Where I have specific questions or issues, they will appear in ***BOLD ITALICS***. These issues will be removed from the document as they are addressed before the final approved document.

2. Project Plan

This section defines the project plan for this document. All known planning and resource issues are discussed in this section.

2.1 Project Team

The following team members will be involved in production of this product.

2.1.1 Software Development

Bob Sigmon
Lena Pavlovskaia
Allan Moluf
Jack Schabel
Tim Lee
Eran Landau
Greg Brown

2.1.2 Quality Assurance

2.1.3 Marketing

2.1.4 Technical Support

2.1.5 Sales

2.2 Schedule

The following project schedule will be employed for these releases:

- [redacted] Architechture/Functional Spec complete, reviewed and ready to go
- [redacted] Tech Specs (Class diagrams, etc.) complete and reviewed (Bob, Lena, Eran)
- [redacted] Arctos Server, Carousel Explorer, Alert Manager, Adminstrator Betas
- [redacted] Ligos Interface
- 12/17/1999 Mux Interface Beta
- 12/20/1999 System testing and bug fixing begins.
- 12/31/1999 System complete

Other milestones:

- [redacted] Hardware available and Software installed
- 12/20/1999 Test Plan
- 12/27/1999 Installation, backup and maintenance documents/scripts

3. Project Requirements

3.1 Functionality

Arctos is an HTML slide show generator that broadcasts on digital MPEG II channels.

1. Must be scalable and able to support at least 20 – 30 carousel channels.
2. Carousel intervals must have a minimum update interval to prevent congestion on the system. Content that is updated at too high of a rate will be dropped.
3. Content will be either local or remote.
4. The channels may be offered in a tiered fashion whereby all digital customers receive a "basic" level of services and have the opportunity to upgrade to a premium level (or levels).
5. Must be able to integrate with either a national or local headend.
6. Consume less than 2MHz (per 20 – 30 channels) on the cable system.
7. Channel content will be developed using any technology supported by Internet Explorer
8. System modifications and status functionality should be accessible through the WEB client.
9. Must have an up time of 99.95%.
10. Must have the ability to set channel parameters referenced in ChannelInterfaceToMUX.doc
11. Notification of changes to HITS virtual channel numbers will by synchronized (Method TBD)
12. The system will send alarms and alerts through an SNMP interface to the head-end(s)
13. Administration features will be protected to the content providers.

Issues:

1. Is there other PSI information neede in the stream.
2. What abou PID 0, I, return PID, etc.

4. Installation

4.1 Initial Installation and Setup

ICTV INC will do installation. The servers will be set up at the home office and delivered in a state ready for 'plug in'. Power (and UPS), network connection to the Internet (and firewall), and connection to the cable system (HITS), is expected to be provided by the customer. The customer will also provide IP addresses (3). Access to the Administration pages will be through the Internet via a browser and system messages will be available through SNMP.

4.2 Maintenance

ICTV INC will provide software maintenance. In order to keep our targeted uptime, the secondary server will be 'disconnected' from the cluster and the software upgrades will be placed on the secondary server. After a successful install, the secondary server will be brought back into the cluster and the primary server will be brought down, forcing the secondary server to become the primary server with the new software. Then maintenance will then be applied to the other server and it will be brought back on line as the secondary. The current version of the software will be accessible via the Admin Pages. At some point, maintenance may be provided remotely, but not initially. SNMP messages will be sent notifying the interested parties that maintenance is being performed.

5. Design Specifications

5.1 Hardware and Software

The Arctos project will be comprised of two Windows NT 4.0 SP6 or later Systems, one of which is a primary server and one a secondary server

Each server will have the following hardware configuration:
1. Dual Pentium PIII 600 processor (or faster)
2. 10 GB Hard drive
3. 512 MB RAM
4. AGP Voodoo 3000 Graphics card
5. Vivid 2-Channel QAM (Could be De-Hi also)
6. CD-ROM
7. 100MPS Network card
8. 1.44 Floppy Disk
9. Zip drive for backup

Each Server will have the following software configuration:
1. Windows NT4.0 SP6
   a. TCP/IP
   b. WLBS (Load Balancing)
   c. SNMP
2. IIS 4.0
3. IE5 5
4. MPEG encoder (Ligos Software/GI Hardware)
5. ICTV Digital Mux Software
6. ICTV Arctos Software
   a. Arctos Server
   b. Carousel Explorer
   c. Alert Manger
   d. Administrator
7. Plug-ins for displaying content including but not restricted to Flash, RealVideo, QuickTime, etc.

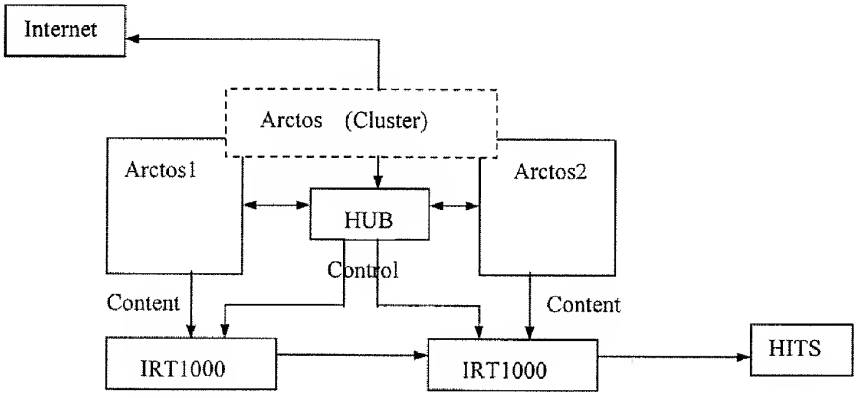
These servers will be connected within their own local network and also to the internet through the head end topology.

5.1.1 Environment

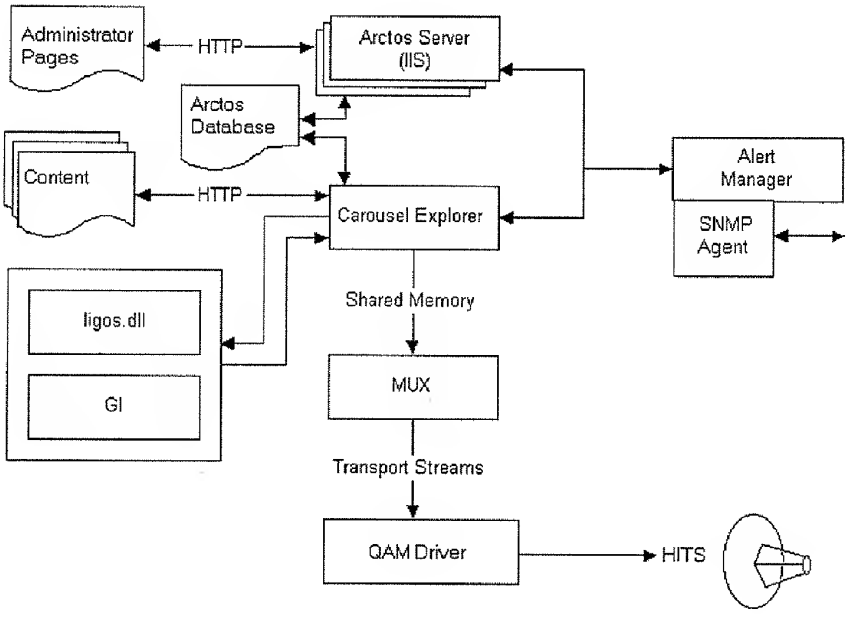
The software environment will consist of Windows NT 4.0 SP6 with Load Balancing software (wbls). The code will be written in C++/Java with administration tool(s) available as HTML/XML forms from a Browser. COM/DCOM will be used to communicate between the services and processes within each server. TCP/IP will be used as the main transport protocol. IIS (Internet Information Server) will be used as the web server. SNMP will be the 'alert' mechanism so that any remote console can view server status at any time. The machines will be 'clustered' together using the load balancing option on NT. They will share a virtual address, which will be the well-known address of the system. Each server will also have individual addresses, although not well known, so that they can be addressed directly. One will be designated as a primary server and one as a secondary server, although this will be only an initial designation and could change over time (see Failsafe section). All parts of the architecture will be a duplicate on each machine, and functionality will be an exact copy.

MPEG Encoding will done by the Ligos software package (or the GI Encoder hardware), and the MUX software will be the same product used in our digital interactive system. Each piece of software will maintain it's 'Version' number and will return it when asked. Later this can be used for software maintenance and troubleshooting.

5.1.2 Arctos Hardware/Network Architecture



5.1.3 Arctos Software Architecture



5.2 Arctos Server

Arctos Server is actually MS IIS 4.0. It controls all outside client access to Arctos.

5.2.1 Functionality

1. Security
   IIS Internet Manager performs authentication for the client.
2. Web Server functions
   Default IIS WEB site will have virtual paths to the Arctos directories where the actual content is stored.
3. CGI programs executed by Arctos Server:
   1. Admin Pages\Channel Map CGI
   2. Admin Pages\System Status CGI
   3. Event Log Viewer CGI

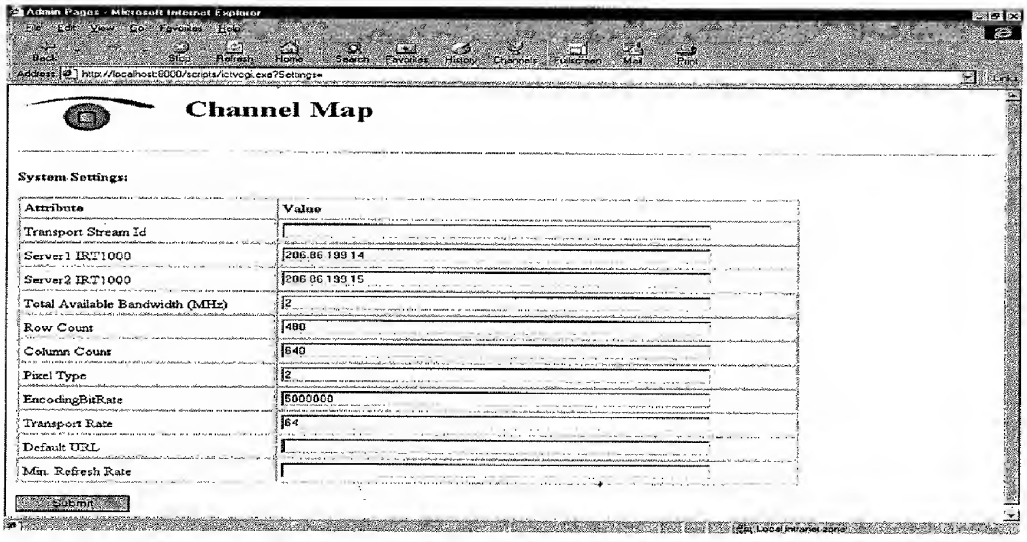
5.3 Carousel Explorer

The Carousel Explorer will be responsible for running the carousel scripts (HTML, XML...) and forwarding the captured sercen shots to the MUX. In addition, Carousel Explorer will be the console application for Arctos and will include functionality for system maintenance and logging.

5.3.1 User Interface

Carousel Explorer will be an MDI application where there is a document and view for every channel. The user interface will have the following components (see image below):

1. All the views will be displayed in Carousel Script Queue position. Each view will contain a browscr control window for displaying the URL.
2. The status bar will contain current system information and distinguish between the primary and backup servers.
3. The Channel Map Frame will provide current status information on cvery channel. The channel map can also be modified through the Admin CGI in a browser control the will be displayed over the Channel Map Frame.
4. The Alert / Log Frame will provide status information as well as system alerts.
5. The Main Toolbar will provide access to various system maintenance and management dialog boxes. These dialog boxes will be displayed over the Alert/Log frame.



5.3.2 Interfaces

1. Arctos Admin CGI will maintain the Channel Map database (See section 5.7.3 for database file format). Carousel Explorer will setup a file change notification and perform the changes whenever the database file changes.

2.Carousel Explorer will communicate with the MUX through dll function calls.
Definition of the interface between the Arctos Carousel process and the shared memory input to the Mux. The Carousel process is concerned with creating and controlling shared memory input channels to | the Mux and with supplying image bitmaps for processing by| the Mux.
The Mux is responsible for converting the bitmap into MPEG and sending the resulting frames according to how the | channel is currently configured.

Carousel-MUX data specifications:

Prefix string A three letter zero-terminated string used to identify a group of related channels. eg. "Bis".
Valid Range: any ASCII letters.
Default Value: "AAA"

Channel int number used to identify a specific channel in a group defined by the prefix.
Valid Range: 1 to 4,095
Default Value: User must specify

Prog int Specifies the program ID number to be used for our PAT/PMT
Valid Range: 1 to 65,535
Default Value: User must specify

Bit Rate float The bitrate in bits per second that should be used for the video elementary stream. This controls the image quality and refresh frequency of the MPEG frames that are transmitted.
Valid Range: 10,000 - 10,000,000
Default Value: 50,000

Hz float Video frames per second: must be one of these values -(23.976, 24, 25, 29.97, 30, 50, 59.94, 60).
Valid Values: 23.976, 29.97
Default Value: 29.97

PID int The PID to use for the video ES in the mux output. Use -1 to dynamically assign a PID for the channel.
Valid Values: 16 - 8190
Default Value: User must specify

PMT int The PID to use for the program's PMT in the mux output. Use -1 to dynamically assign this value.
Valid Values: 16 - 8190
Default Value: User must specify

rowCount long Number of pixel rows in each input bitmap.
Valid Values: 480
Default Value: 480

colCount long Number of pixel columns in each input bitmap.
Valid Values: 640
Default Value: 640

PixelType int Some number that identifies the pixel type so that the bitmap format can be interpreted for MPEG encoding.
Default Value: 2 = cmRGB565

Buffer string OUT: On return from 'CmAddChannel()' and 'CmUpdateChannel()' this field holds the address of the buffer where bitmap data should be put for input to the channel.

DataSize string OUT: On return from 'CmAddChannel()' and 'CmUpdateChannel()' this field holds the number of bytes allocated for bitmap data at 'Buffer'.

Dll will have the following methods:

```
int CmAddChannel( CmChannelSpec* );
int CmRemoveChannel( char*, int );
int CmResumeChannel( char*, int );
int CmResumeAll();
int CmSuspendChannel( char*, int );
int CmSuspendAll();
int CmUpdateChannel( CmChannelSpec* );
```

5.3.3 Carousel Scripts

The carousel scripts are html pages that can be updated using one of the following techniques:

1. HTML Timer – In this case the HTML page must contain a timer that tells the page to either update or navigate to a new URL.
2. AutoRefresh Timer – The channel will be setup with one URL that is refreshed at a certain interval. This technique should be used if there is content consists of one html page that is changed frequently.
3. AutoCapture Timer – The channel will be setup with one URL that is captured at a certain interval. This technique should be used it the html page contains an ActiveX component that changes without sending a navigate change event.

Procedure for capturing new frames:

1. New page is loaded into browser window or AutoRefresh or AutoCapture timers are fired
2. The Channel Document adds itself to the update queue.
3. A timer running in the main window checks this queue at a certain interval. Whenever a new document is found it's view is brought to the top of the display (Carousel Explorer can only capture windows that are visible in the display).
4. A second timer gives the window enough time to redraw.
5. The view image is captured and passed to the MUX.

Error handling:

In the event that a page was not loaded successfully, Carousel Explorer will follow this procedure:

1. Carousel Explorer will attempt to display the ErrorURL that was specified with the channel definition.
2. If this page is not found or could not load, the default ErrorURL page (which is local) will be displayed.
3. The channel will transition to an error state and send an alert to the alert manager.
4. The channel will remain in this state until the problem is fixed.

5.3.4 Channel Map Frame

The channel map frame will display status information for each channel and provide an interface for modifying the channel map.

The channel map list will provide the following information:

1. Items will be sorted by status and channel number. Channels with errors will be at the top of the list
2. Each item will be color coded to indicate the status.
   i) Blue – Channel is active
   ii) Grey – Channel has been deactivated
   iii) Red – Error
3. The list will include these columns
   i) Channel status icon (Active, Inactive, Error)
   ii) Channel Number
   iii) Channel Name
   iv) Start URL

The channel map toolbar will provide buttons for Adding, Removing, Updating, Activating and Deactivating a channel. To ensure that only one process maintains the database Carousel Explorer will send commands to modify the channel map to the Admin CGI.

5.3.5 Alert / Log Frame

The Alert / Log frame will display system status information or system alerts. The list will provide the following information:

1. Items will be color coded to indicate severity
   i) Red – severe error

ii) Orange – warning
iii) Yellow – informative event
2. Columns
i)
ii) Time
iii) Channel name or system component name
iv) Event text

5.3.6 Status Frame

The status frame will provide the following information:
1. Machine name (Arctos1 or Arctos2)
2. The background color of the frame will indicate whether this is the primary or backup machine. In the event of a sever error the frame background will blink
3. System Uptime

5.3.7 Main toolbar

The main toolbar will provide access to the following functionality:
1. System configuration
2. Shutting down the system and switching to the backup machine
3. MUX configuration

MPEG Encoder

There are two possibilities for generating an IFrame for the screen shots.

1. Use Ligos to compress the image data in software
2. Use GI to compress the image data in hardware

5.3.8 Functionality

1. Receives a bitmap image from the Carousel Explorer and returns an encoded Iframe to the Carousel Explorer

5.3.9 Interfaces

5.4 MUX

The mux is responsible for continuously broadcasting an MPEG stream for every channel.

5.4.1 Functionality

1. Continuously broadcast channels at 30 frames per second.
2. 1 thread per channel
3. Gets IFrame updates from the shared memory files produced by the Carousel Explorer
4. Comes up in standby broadcasting null packed to Vivid mux.
5. Creates shared memory and locks for mux control.
6. Waits for channel create or other control messages.
7. Creates shared memory and locks for new channel(s)
8. Acknowledges that the new channel is created and ready to use.
9. Channel messages includc:
   a. Pause and Resume
   b. Channel content (pictures)
   c. Close channel (delete) and removes locks and shared memory for that channel.
10. Mux Activate
    a. disables any other mux output to IRT1000
    b. enables its own IRT1000
    c. start placing packets into output stream (which is running steadily)
11. Gcncratc PAT's and PMT's for crcated channels.
12. System configurablc bandwidth throttle.
13. Schedule Iframe between different channels.

5.4.2 Interfaces

1. Shared Memory
   a. Channel controls
   b. Mux controls (add channels/start stop mux, query status and version number)

5.5 Alert Manager

The alert manager will be responsible for monitoring the status of the system and interfacing with the monitoring software at the headend. The alert manager will consist of an interface dll and the SNMP Extendible Agent with an Arctos specific MIB. Alerts in this case also retrieve information, so they are more than just alerts in the normal sense.

5.5.1 Functionality

1. Each piece of software must 'register' with the Alert Manager as it starts
2. Registration implies 'up' status.
3. Respond to SNMP requests from the monitoring software at the head end and the internal Arctos Software.
   a. Get
   b. GetNext
4. Respond to SET requests from the Arctos Servers and Carousel Explorer. The head-end will request SET alerts through the administrator, not through SNMP requests directly. These will be requested through the DLL interface.
5. Forward/Return the appropriate alerts.
6. Types of alerts and disposition:
   a. Event – Places alert into Event Log
   b. Status – Replies with system status
   c. Version- Replies with the software version(s)
   d. Email - Sends email
   e. Reboot – Reboots server(s)

5.5.2 Interfaces

The AlertManager interface will be callable DLL entry points, which will handle the request or dispatch it to the SNMPAgent or other appropriate processes.

1. Register (const char* name, const char* version)
2. Event(const char* event)
3. int GetServerStatus(const int serverNumber)
4. int GetSoftwareStatus(const char* name)
5. const char*GetSoftwareVersion(const char* name)
6. SendEMail(const char* message)
7. RebootServer(const int serverNumber)
8. SetServerStatus(const int serverNumber, const int status)
9. SetSoftwareStatus (const char* name, const int status)

5.6 Administrator tools

The administrator tools will be presented as HTML pages and will allow the addition, deletion, update and status of channels on the system.

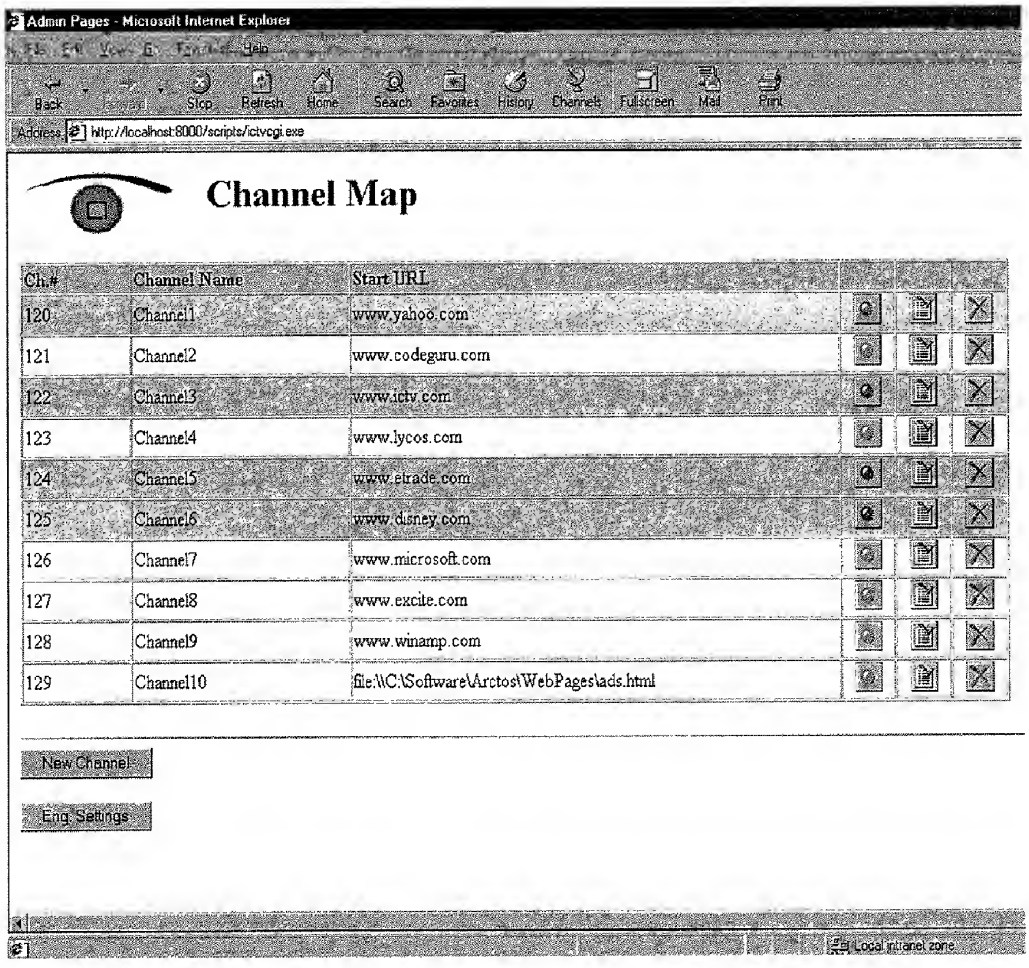
5.6.1 Functionality

1. Registers with Alert Manager
2. Administration processes:
   - i) Adding channels
     - a) The Admimistrator uses the WEB Client connected to the ARCTOS Server on the primary machine telling it to add a channel.
     - b) ARCTOS Server Updates Database
     - c) The primary machine notifies the secondary machine of the change.
     - d) Carousel Explorer picks up the new channel info.
     - e) Carousel Explorer creates a new window and positions it in the new channel position.
     - f) Carousel Explorer notifies the MUX of the new channel
     - g) Load the first page and begin handling the page transitions.
     - h) Return a either a success or failure HTML page to the content provider.
   - ii) Removing
     - a) The Administrator uses the WEB Client connected to the ARCTOS Server on the primary machine telling it to remove a channel.
     - i) ARCTOS Server Updates Database
     - b) The primary machine notifies the secondary machine of the change.
     - c) Carousel Explorer picks up the change flag to remove the channel.
     - d) Carousel Explorer removes the child browser window for that channel.
     - e) The Carousel Explorer tells the MUX to remove the channel
   - iii) Updating
     - a) The Administrator uses the WEB Client connected to the ARCTOS Server on the primary machine telling it to change either the start URL.
     - b) ARCTOS Server Updates Database
     - c) The primary machine notifies the secondary machine of the change.
     - d) Carousel Explorer picks up the updated channel info (including the new URL for the specified channel)
     - e) Content Refresh
   - iv) Status
     - a) The Administrator uses the WEB Client connected to the ARCTOS Server on the primary machine requesting a status page.
     - b) The ARCTOS Server generates a status page that contains
       1. Channel number
       2. First URL
       3. Current URL
       4. Uptime
       5. Last time page was displayed

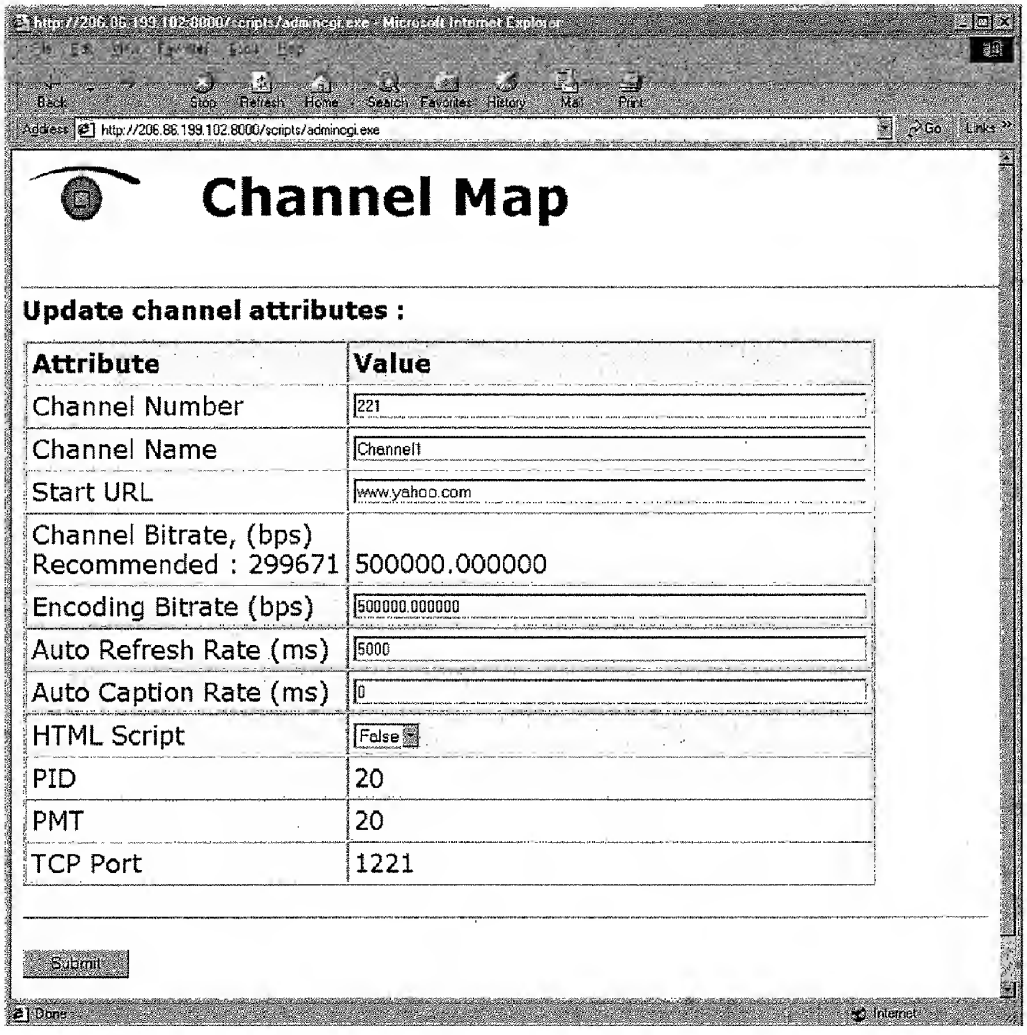
   c). Store reliability records.

5.6.2 User Interface

The user interface for the Administration pages will be HTML forms accessible through a browser. The pages will be able to be 'branded' by the customer.
ICTV Admin Pages\Channel Map is an application that provides a user interface to the channel map.
User can add, remove, update, enable, disable channels.



5.6.3 Carousel Explorer Interface



Channel map data specifications:

Channel # int
Valid Range: 1 to 4,095
Default Value: User must specify
Channel Name string A mnemonic name for a channel. Maximum length - 100 characters
Start URL string Maximum length - 100 characters.

Channel Bit Rate float The bit rate in bits per second that should be used for the video elementary stream. This controls the image quality and refresh frequency of the MPEG frames that are transmitted.
Valid Range: 10,000 – to a limit, calculated from the System settings
Recommended Value: calculated

Encoding Bit Rate float Video frames per second: must be one of these values - (23.976, 29.97).

PID int The PID to use for the video ES in the mux output.
Valid Values: 16 - 8190
Default Value: User must specify

PMT int The PID to use for the program's PMT in the mux output.
Valid Values: 16 - 8190
Default Value: User must specify

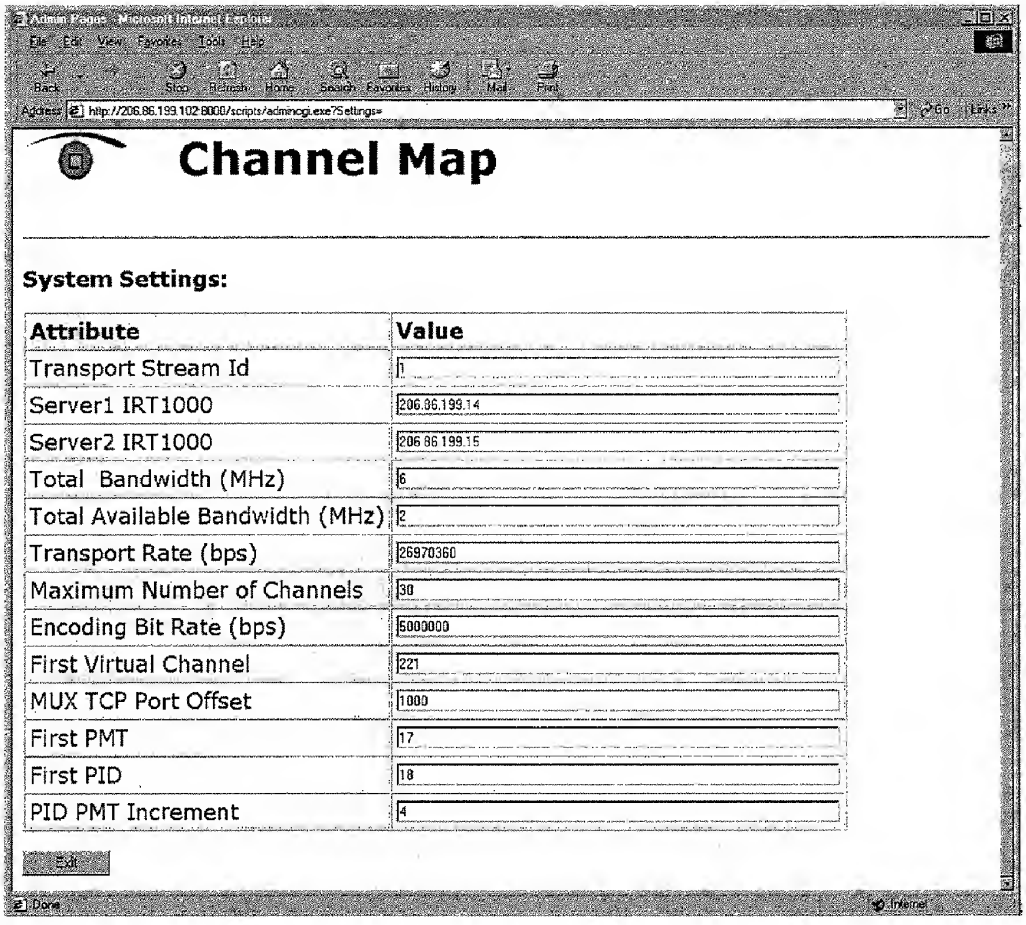
AutoRefresh int The channel will be setup with one URL that is refreshed at a certain interval.

AutoCapture int The channel will be setup with one URL that is captured at a certain interval. (either AutoRefresh or AutoCapture must be 0)

HTML Script bool the HTML script or link

Mux Port int TCP port to be used in Mux for that channel

5.6.4 System Settings Interface



Transport Stream Id int
Server1IRT1000 int
Server2IRT1000 int
Total Bandwidth int (MHz)
Default Value: 6
Total Avail. Bandwidth int (MHz)
Default Value: 2
Transport Rate int (Bps)
Default Value: 26970360
Max # of Channels int
Default Value: 30
EncodingBitRate double The bitrate in bits per second that controls the image quality setting in the MPEG encoder.
Valid Range: 1,000,000 - 10,000,000

	Default Value: 5,000,000
rowCount	long Number of pixel rows in each input bitmap.
	Valid Values: 480
	Default Value: 480
colCount	long Number of pixel columns in each input bitmap.
	Valid Values: 640
	Default Value: 640
defaultSystemURL	string Number of pixel columns in each input bitmap.
minRefreshRate	long sec

5.7 Failsafe

Windows NT clustering will be used to set up a primary/secondary server relationship. We will use the cluster address to 'get to' the server(s). NT clustering will then pass the traffic to the correct server. When a server fails, it is taken out of the cluster and all traffic is transferred to the other member(s) of the cluster with no interruption of the traffic flow. This will require NT Server 4.0 SP6 with the load balancing option (wlbs). You can still access the individual servers through their unique IP addresses.

This system will be set up as a cluster in which there is a primary and a secondary server. When the primary server crashes or fails, the secondary server must be able to take its place, in a reasonable amount of time. If any portion of the software fails, the server must be able to recover without loss of service. Each system must have it's own copy of the software and data, which must be synchronized. When the problem is corrected, the server that became the primary server stays as the primary server and the recovered server becomes the secondary server.

5.7.1 Startup and Operation

1. One machine is initially setup as a primary server while the other is setup as a secondary server
2. Primary machine notifies the logging service that it is up and is the primary
3. The primary machine loads the channel carousel and begins broadcasting
4. The secondary server realizes it is the secondary and requests file synchronization
5. Secondary server notifies the logging service it is up and is the secondary
6. When changes to the channel carousel are made, the secondary is notified (see 5.7.1)
7. Periodic file synchronization will be requested from the secondary.

5.7.2 Failures scenarios:

5.7.2.1 Primary server fails

1. The secondary server realizes that the primary server failed
2. Alerts are sent to the alert manager
3. The secondary should have all the channels loaded and setup
4. The secondary server sets itself as a primary server
5. T he ArctosServer module is an NT service, it will be already running and ready to take over
6. Broadcasting resumes with only minor interruptions in service
7. Human intervention may be required to fix the failed server.

5.7.2.2 Secondary server fails

1. The primary server realizes the secondary server has failed
2. Alerts are sent to the alert manager
3. The primary continues broadcasting
4. Changes are applied and not synchronized
5. Human intervention may be required to fix the failed server.

5.7.2.3 Failed server Recovers:

1. Failed server is recovered and comes back on line and registers as secondary server
2. Alert is sent to the alert manager that the server is back up
3. Server requests file synchronization
4. Changes on primary are now only considered applied after secondary responds.

5.7.2.4 Failed Server Processes

1. Process failures (services or programs or dlls) should be handled by the process itself.
2. Process fails (GPF or other condition)
3. Alert is sent to the alert manager, with as much info as possible
4. Process is restarted. If the process fails again, the secondary server will take over. If the process fails on the secondary server, data will be refreshed. A failure after the data refresh will be fatal. We don't want to get into a catch-22 with ping-pong failures.
5. Human intervention may be required to fix server.

5.7.3 Synchronization

1. If the secondary is up and running, any change is not considered applied until the secondary machine replies that it has applied the change
2. If the secondary is not responding, the change is applied and marked complete. When the secondary machine comes back on-line, it will request file synchronization.

6. Test Plan

The test plan for this project is contained in the *Arctos Test Plan*, document number TP0001.01, by the Arctos Team

7. Test Report

The Test Report section will be completed at the end of the project. It should include the results of the testing process, the number and nature of any outstanding issues or problems, and any recommendations for future improvements in development, testing, or release practices.